Numerical tools for some identification problems in industrial applications

Von der Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktor-Ingenieur (Dr.-Ing.)

genehmigte
Dissertation

von Muhammad Noman Kabir
geboren am 01.01.1977
in Bhanga, Bangladesch

Eingereicht am: 11.07.2007
Mündliche Prüfung am: 01.10.2007

Mentorin: Prof. Dr. Heike Faßbender

Vorsitzender: Prof. Dr. Sándor Fekete
Referentin: Prof. Dr. Heike Faßbender
Korreferenten: Prof. Dr. Matthias Bollhöfer
Prof. Dr.-Ing. Manfred Krafczyk

2007

Declaration

I hereby declare that this dissertation is entirely the result of my own work except where otherwise mentioned. I have only used the resources given in the list of references.

Muhammad Noman Kabir

Abstract

In many industrial applications, models constructed from real problems using empirical or physical laws are used for control, prediction, error detection, design, or simulation. Models often involve some unknown coefficients (parameters). The unknown parameters of a model have to be estimated prior to solving the model. Parameters can be estimated using the model and the observed (measured) data from the real problem. Parameter estimation problems take (can be reduced to) various forms, i.e. "nonlinear optimization", "nonlinear equations" or "nonlinear least squares" which can be solved by numerical methods. There are many numerical methods, each of which is appropriate for a specific variety of problem.
In this dissertation, Newton's method and its variants (Newton's method with line search/trust-region) for solving parameter estimation problems of small to medium scale are addressed. Newton's method may fail if the Hessian (Jacobian) matrices are singular or indefinite (singular). Newton's method with trust-region can be used to avoid such problem. The search direction in this method can be computed by a sequence of factorizing of the Hessian (Jacobian) matrix with diagonally modified structure where the final modified system turns out to be convex and well-conditioned (regularized).
Since numerical methods are used to solve the problems, certain numerical issues, i.e. underflow, overflow, rounding error and cancellation are introduced. Enhancing the performance of the algorithm is also elaborated.
Implementation techniques of the algorithms are outlined in details. Test results are given to illustrate performances of different variants of Newton's method.
Solution procedures of the following parameter identification problems from the company IAV GmbH, Gifhorn are discussed.

- Identification of geometric tolerances in a sensor wheel;
- Identification of combustion parameters in a diesel engine;
- Identification of reaction parameters in a "Selective Catalytic Reduction" (SCR) catalyst.

Models and their computational heuristics of above identification problems are described. Parameters are estimated using variants of Newton's method with the measurement data of the company. Results from the identifications are presented.

A new trust-region algorithm for the nonlinear system of equations and the nonlinear least squares is presented. The trust-region concept from unconstrained optimization is used to derive the algorithm. Thus the algorithm works well, even if the Jacobian matrices become nearly singular.
Alternative models to the Wiebe function [38] for modeling the combustion process in a diesel engine are proposed since identification of combustion parameters using the Wiebe function is computationally too expensive to be used in the real time operation. Computational time for parameter identification with new model functions is significantly less than that with the Wiebe function. This makes new functions suitable choices to be used for the real time operation.

Acknowledgements

First and foremost, I am profoundly grateful to my mentor Prof. Heike Faßbender for her wise counsel and patient support throughout this dissertation. I thank her for the continuous guidance and advice with which my dissertation has been brought to success. I especially thank my former colleague Dr. Tobius Damm who introduced me to many complicated theories of parameter identification.

This dissertation was supported by IAV GmbH, Gifhorn, Germany. I would like to thank the people from IAV who helped my dissertation in various ways. I am indebted to Winfried Schultalbers and Friedhelm Laubenstein for organizing financial support. I would like to express my gratitude to Thorsten Schmidt for his instruction and encouragement during the tenure of my dissertation. I thank Immo Müller de Vries for his support during learning of measurement of diesel engines and the software development tool ASCET. His suggestions helped me to improve my dissertation. Special thanks go to Dr. Kay-Jochen Langeheinecke for his assistance in learning more about computational fluid dynamics. I benefited from many discussions with him.

Finally I thank all of my colleagues from the Institute of Computational Mathematics and IAV for their direct or indirect assistance and inspiration.

Muhammad Noman Kabir

Contents

Tables and Figures

Tables

Figures

Chapter 1

Introduction

1.1 Motivation

Parameter identification is the estimation of unknown parameters in a model constructed from a real problem. The model can be constructed from empirical as well as physical laws. Parameter estimation is based on the model and the observed (measured) data from the real problem. Unknown parameters are estimated by minimizing the discrepancy between measured output quantities and corresponding quantities obtained by solving the model.

A procedure for parameter identification in real problems can be divided into the following parts:

1. Model determination: The mathematical model is constructed from basic physical laws and empirical considerations of a real problem. The mathematical model may not have any analytical solution or may not be solved by hand computation and therefore we require a computer to perform the task by numerical methods. A numerical method solving a model on the computer most often gives a practical solution to the real problem if the real problem is adequately represented by the model. In fact, when a real problem is mathematically expressed, it is sometimes simplified so that the real problem is represented approximately. When this model is solved by a numerical method, it may be necessary to perform further simplification.

2. Experiment: Measurement is carried out for the real problem and input and output values are sampled and stored.

3. Parameter estimation: Unknown parameters are estimated by minimizing the deviation between the sampled output and the model output. This problem can take (can be reduced to) a form of *optimization*, *system of equations* or *least squares*. The parameters are estimated by solving the problem of corresponding form.

4. Model validation: The correctness of the model and the accuracy of the parameter estimates are evaluated, based on the model fit and physical ranges of parameters.

In the model predictive control (MPC) of a process, we have a parameter identification problem where a mathematical model is used to predict the dynamic behavior of the process (real problem) over a future horizon. The model involves control variables which are the unknown parameters. Control variables are estimated so that the discrepancy between the desired control output (like measured output) and the model predicted control output (model output) is minimal. Computed control variables are used to control the

process. Since MPC is a real time process, the whole procedure is repeated at each control interval [37].

Parameter identification in real problems in many industrial research areas very often exhibit nonlinear behavior, and therefore constructed models are nonlinear. In particular, this dissertation will provide numerical methods to solve the following three parameter identification problems.

1. The *Selective Catalytic Reduction* (SCR) catalyst is currently considered the most effective technology for high nitrogen oxides (NO_x) removal from exhaust gas of a diesel engine using ammonia as a reducing agent. In the SCR catalyst, ammonia is adsorbed in the catalyst and reactions take place between adsorbed ammonia and NO_x to form water vapor (H_2O) and nitrogen gas (N_2). The SCR catalyst has very high conversion efficiencies, typically 85% to 99% for the reduction of NO_x to N_2 [23]. The chemical reaction parameters have to be identified for designing (or predicting from) an SCR catalyst.

   Identification of reaction parameters in an SCR catalyst gives a highly nonlinear least squares problem that is formulated using an SCR catalyst model [23, 39] and measurements of flow at entrance and exit of the catalyst. The physical phenomenon "exhaust gas flow through an SCR catalyst" can be expressed in terms of a mathematical model using *partial differential equations* (PDEs). PDEs are solved by a numerical method on a computer. The physical domain can be discretized into finitely many volume elements and the time domain can be discretized into finitely many time steps. PDEs can be converted into a system of algebraic equations using the *finite volume method* (FVM) for volume elements, and an *implicit* method for time steps. Boundary conditions obtained from the data at entrance of the catalyst are set in this system. The system is solved for the quantities "temperatures" and "concentrations of species" for all discrete volume elements over time steps. A nonlinear least squares problem is formulated using this simulation model and the data at exit of the catalyst. By solving this problem, we can estimate the reaction parameters.

   Higher accuracy of the solution can be obtained using finer volume elements and finer time steps. The number of volume elements and time steps are restricted by the computer memory and time frame within which the solution is expected.

2. A simplified mathematical model of the combustion process in a diesel engine is the *Wiebe function* [38] which is widely used in application. The model has three combustion parameters: *crank angle at the start of ignition*, *total amount of heat release* and *combustion duration* which can be identified and used to control the speed, efficiency, noise and exhaust emission. These parameters can be estimated by minimizing the deviation between the pressure signal from measurements and the pressure derived from the combustion model that can be formulated as a nonlinear least squares problem. The parameters have to be computed fast enough with respect to engine speed so that estimated parameters can be used in the real

time. A simplified model of the combustion process can be considered to save the computational time.

Investigation shows that the computational time to determine the combustion parameters using the Wiebe function is too high to be suitable in the real time operation. That is why alternative models are proposed which are cheap in computing combustion parameters and provide suitable choices for the real time application.

3. Modern diesel fuel injection system (common rail and pump-nozzle-units) requires the reference of crank angles, engine speed, load etc. to estimate the two parameters: *crank angle at the start of fuel injection* and *amount of fuel to inject* for a diesel engine. Measurement of engine speed of a combustion engine is carried out by sensor wheels mounted on the crankshaft. Crank angles can be estimated from the measured engine speed. But geometrical tolerances of a sensor wheel result in systematic errors in the measurement. To get the corrected speed, we have to determine these errors due to geometric tolerances. These geometrical tolerances can be estimated by solving a nonlinear system of equations which is formulated using a suitable physical model of kinetic energy balance of the crankshaft and the measurement of engine speed [15].

Since we employ a computer to solve the problems, we need to consider certain issues, e.g., underflow, overflow and cancellation. As computer can store finite precision for numbers, a problem sometimes cannot be solved beyond a certain precision. However, it most often provides a solution that is accurate enough for practical applications. Underflow and overflow may cause failure to an algorithm. Those can be avoided through proper scaling. Cancellation may cause loss of accuracy leading to a wrong result. Cancellation can be avoided by rearranging the algorithm.

Increasing computer memory and speed year on year gives a huge benefit in terms of required time and accuracy for numerical simulation. Numerical simulation tools are massively used in industry and at the present time many expensive experiments are replaced by numerical simulations. Sometimes, a computer model of a nonlinear system is comprised of complex and time-consuming numerical simulation. Designing a fast algorithm is very important to obtain the solution within reasonable time. Enhancing the speed of the algorithm is utmost significant for problems in the real time operation.

1.2 Research problem

IAV requires a new, fast, stand-alone and optimized software of parameter identification for their automobile controllers that use “ETAS rapid prototyping hardwares”. The hardware supports ASCET (a modeling and development tool for automotive embedded system) which includes a C-like language ASCET-C. The typical problems of parameter identification that are intended to be solved by the software are nonlinear and of small to medium scale. Parameter estimation problems take the following forms

1. nonlinear optimization,

2. nonlinear least squares,

3. nonlinear system of equations.

Therefore, the software should include the solvers for above forms.

The software has to be developed in C using fast and robust algorithms and then has to be converted in ASCET-C for the real time operation. The software is to be tested with the problems mainly arisen from the area of diesel engines.

There is a number of commercial and noncommercial softwares for solving problems of above three forms available at the present time. Most of available C codes are complicated and call different Fortran routines. Moreover, they may not be optimized for a different hardware. Therefore, it necessitates to develop a new code or to modify an existing code.

Due to the copyright restriction, we will only present the design and tests of algorithms in C language but not in ASCET-C.

1.3 Outline of the report

This dissertation intends to present all the theories, algorithms required to design and implement a software for solving nonlinear unconstrained optimization, nonlinear equations and nonlinear least squares.

Since problems will be solved numerically, it is very important to know how the computer represents data, how it calculates a value. Chapter 2 is a detail description of fundamentals of numerical computing. This knowledge is very important prior to implement an algorithm. Underflow, overflow, rounding error and cancellation are described with examples in this chapter. In control systems the algorithm must work fast so that the output of an algorithm can be used in real time. This chapter describes how to increase the speed of an algorithm.

Chapter 3 describes the nonlinear unconstrained optimization. Chapter 4 covers the nonlinear least squares and nonlinear system of equations. These two chapters contain various solution methodologies with algorithms and technical details.

Chapter 5 presents implementation details of algorithms. Various test problems were investigated by implemented algorithms (software). Chapter 6 gives test results from the software.

Chapter 7, 8, 9 provide three parameter identification problems (as mentioned in section 1.1) from diesel engines and their solutions using the software.

Lastly chapter 10 is a summary and conclusion of the whole work.

1.4 Notations and fundamentals of analysis

We will use $\mathbb{R}$ to denote the set of real numbers. $\mathbb{R}^n$ and $\mathbb{R}^{m\times n}$ will express the set of n-dimensional real vectors, and the set of m by n real matrices, respectively. We will use the **subscript index** $\boldsymbol{i}$ to indicate the ith component of the associated vector. The **subscript index** $\boldsymbol{i\,j}$ will be used to denote (i, j)th element of the associated matrix. The

notation **diag**(d) will be used to denote a diagonal matrix with diagonal elements d_i with $i = 1, 2, \cdots$. The superscript T will denote the transpose of the associated quantity. A vector $x \in \mathbb{R}^n$ will be considered as a column vector. The **scalar product** of two vectors $x, y \in \mathbb{R}^n$ is given by $\sum_{i=1}^n x_i y_i = x^T y$. The **vector** p**-norm** for $p = 1, 2, \cdots$ is defined by

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}. \tag{1.1}$$

The **Euclidean norm** is given by

$$\|x\|_2 = \sqrt{x^T x} = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{1.2}$$

We will use $\|x\|$ to indicate any norm of x. The notation $\angle(x, y)$ will be used to imply the angle between two vectors $x, y \in \mathbb{R}^n$. A **ceil function** of a real number y is the function which returns the smallest integer not less than y and will be denoted by $\lceil y \rceil$.

The **gradient** (vector of partial derivatives) of a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ will be denoted by $g, \nabla f$ or f', which is given by

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n. \tag{1.3}$$

The **Hessian** (matrix of second partial derivatives) of a twice-continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ will be denoted by H or $\nabla^2 f$, which is given by

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_n \partial x_n} \end{pmatrix} \in \mathbb{R}^{n \times n}. \tag{1.4}$$

If $c : \mathbb{R}^n \to \mathbb{R}^m$ is continuously differentiable, then

$$A(x) = \begin{pmatrix} \frac{\partial c_1(x)}{\partial x_1} & \cdots & \frac{\partial c_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial c_m(x)}{\partial x_1} & \cdots & \frac{\partial c_m(x)}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n} \tag{1.5}$$

will denote the **Jacobian** matrix of c.
An infinite sequence of vectors x_k has a **limit** x^* if for all $\epsilon > 0$, there exists an integer $k(\epsilon)$ such that for every $k \geq k(\epsilon)$

$$\|x_k - x^*\| < \epsilon. \tag{1.6}$$

When a sequence has a limit, the notation

$$\lim_{k\to\infty} x_k = x^* \tag{1.7}$$

will be used. This sequence is said to **converge** or to be a **convergent sequence**. A point $\overline{x}$ is called a **point of accumulation** of an infinite sequence x_k if there exists an infinite sequence of integers $k_1, k_2, k_3 \cdots$ such that

$$\lim_{i\to\infty} x_{k_i} = \overline{x}. \tag{1.8}$$

A sequence x_k is called **bounded** if there is a real number $C > 0$ such that

$$\|x_k\| \leq C \qquad \forall k. \tag{1.9}$$

If an infinite sequence of vectors x_k is bounded and if it has only one point of accumulation, then the sequence converges and has a **limit**. A set $S \subset \mathbb{R}^n$ is called **closed** if it contains the limit of every convergent sequence of points formed from S. A set $S \subset \mathbb{R}^n$ is called **bounded** if there is a real number $C > 0$ such that

$$\|x\| \leq C \qquad \forall x \in S. \tag{1.10}$$

A set which is closed and bounded is called **compact**. The **neighborhood** of a point x^* will be denoted by $\mathbb{B}_\epsilon(x^*)$, which is defined by

$$\mathbb{B}_\epsilon(x^*) \quad := \quad \{x \in \mathbb{R}^n : \|x - x^*\| < \epsilon\}. \tag{1.11}$$

A **level set** of x_0 is defined by

$$N_f(x_0) := \{x \mid f(x) \leq f(x_0)\}.$$

A set S is open if for every $x \in S$, there is an $\epsilon(x)$ such that $\mathbb{B}_\epsilon(x)$ is entirely contained in S. A matrix $M \in \mathbb{R}^{n\times n}$ is called

1. **positive semi-definite** if $x^T M x \geq 0$ for all nonzero $x \in \mathbb{R}^n$,
2. **positive definite** if $x^T M x > 0$ for all nonzero $x \in \mathbb{R}^n$,
3. **negative semi-definite** if $x^T M x \leq 0$ for all nonzero $x \in \mathbb{R}^n$,
4. **negative definite** if $x^T M x < 0$ for all nonzero $x \in \mathbb{R}^n$,
5. **indefinite** if there exist nonzero vectors $x, y \in \mathbb{R}^n$ such that $x^T M x < 0$ and $y^T M y > 0$.

The **matrix p-norm** for $p = 1, 2, \cdots$ is defined by

$$\|M\|_p = \max_{x \text{ s.t. } \|x\|_p=1} \|Mx\|_p, \tag{1.12}$$

where $\|x\|_p$ is the vector p-norm. The **matrix 1-norm** $\|M\|_1$ is defined by

$$\|M\|_1 = \max_j \sum_{i=1}^{n} |M_{ij}| = \text{ maximum absolute column sum norm.} \tag{1.13}$$

The **matrix ∞-norm** $\|M\|_\infty$ is defined by

$$\|M\|_\infty = \max_i \sum_{j=1}^{n} |M_{ij}| = \text{ maximum absolute row sum norm.} \tag{1.14}$$

A function $J : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ is **bounded** if there exists a real number $C > 0$ such that

$$\|J(x)\| < C \qquad \forall x \in \mathbb{C}^n. \tag{1.15}$$

A function $J : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ is said to be **Lipschitz continuous** if there exists a constant $\beta_L > 0$ such that

$$\|J(x_1) - J(x_2)\| \leq \beta_L \|x_1 - x_2\| \tag{1.16}$$

for all $x_1, x_2 \in \mathbb{R}^n$. If J is Lipschitz continuous, then it is continuous in x. The converse is not necessarily true. However if J has bounded partial derivatives in x, then it is Lipschitz continuous.
We will use the symbols $O(h^k)$ and $o(h^k), k \in \mathbb{N}$ where $\mathbb{N}$ is the set of natural numbers. For $O(h^k)$, we write

$$g(h) = O(h^k) \text{ for } h \to 0, \text{ and}$$
$$\limsup_{h\to 0} \frac{\|g(h)\|}{|h|^k} < \infty,$$

and for $o(h^k)$, we write

$$g(h) = o(h^k) \text{ for } h \to 0, \text{ and}$$
$$\limsup_{h\to 0} \frac{\|g(h)\|}{|h|^k} = 0,$$

with a function $g : \mathbb{R} \to \mathbb{R}^n$.

Definition 1.1 ***Rate of convergence [31, p. 28–30]***

1. *A sequence $x_k \in \mathbb{R}^n$ converges **Q-linearly** to x^* with a rate $\kappa \in (0, 1)$ if there is an $l \geq 0$ such that,*

$$\|x_{k+1} - x^*\| \leq \kappa \|x_k - x^*\| \quad \forall k \geq l.$$

2. *A sequence $x_k \in \mathbb{R}^n$ converges **Q-superlinearly** to x^* if $x_k \to x^*$ and*

$$\|x_{k+1} - x^*\| = o(\|x_k - x^*\|) \quad \text{for } k \to \infty.$$

With this definition the following form holds true:

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \to 0.$$

3. *A sequence $x_k \in \mathbb{R}^n$ converges* ***Q-quadratic*** *to x^* if $x_k \to x^*$ and*

$$\|x_{k+1} - x^*\| = O(\|x_k - x^*\|^2) \quad \text{for } k \to \infty.$$

With this definition there exists a $C > 0$ and $l > 0$ such that

$$\|x_{k+1} - x^*\| \leq C\|x_k - x^*\|^2 \quad \forall k \geq l.$$

4. *A sequence $x_k \in \mathbb{R}^n$ converges* ***R-linearly, R-superlinearly or R-quadratically*** *to x^* if there exists a sequaence $\nu_k \in \mathbb{R}$ which converges to 0 with Q-linear, Q-superlinear or Q-quadratical rate, respectively such that*

$$\|x_k - x^*\| \leq \nu_k \quad \forall k \geq 0.$$

The prefix Q stands for "Quotient" and is used to differentiate from R(Root) orders of convergence.

We represent the **subscript index k** to denote the quantities evaluated at the kth iteration. For example, x_{ik} implies the ith component of x evaluated at the kth iteration, and the Jacobian matrix A_k indicates the Jacobian matrix $A(x_k)$. The Hessian matrix H_k indicates the Hessian matrix $H(x_k)$.

1.5 Optimization terminologies

We consider the following continuous nonlinear optimization problem [40].

$$\min_x \quad f(x) \tag{1.17}$$
$$\text{subject to} \quad x \quad \in Z, \tag{1.18}$$

where $Z \subset \mathbb{R}^n$ is the **feasible region**, $x = [x_1, x_2, \cdots, x_n]^T$ is called the **optimizing parameter**, and $f : \mathbb{R}^n \to \mathbb{R}$ is the **objective function** which is assumed to be continuously differentiable. $x \in Z$ is called the **constraints**. The maximization problem can be transformed into a minimization problem simply replacing f by $-f$. The feasible region Z can be described in terms of equalities and inequalities.

$$Z = \begin{cases} c_i(x) = 0, & i \in \mathcal{E} = \{1, 2, \cdots, m\}, \\ c_i(x) \leq 0, & i \in \mathcal{I} = \{m+1, m+2, \cdots, m+l\}, \end{cases} \tag{1.19}$$

where $c : \mathbb{R}^n \to \mathbb{R}^{m+l}$ are continuously differentiable functions. A function $c_i(x) = 0, i \in \mathcal{E}$ is called an **equality constraint** and the function $c_i(x) \leq 0$, $i \in \mathcal{I}$ is called the **inequal-**

ity constraint. The problem consisting only of (1.17) is called the **unconstrained optimization** problem, while the problem consisting of (1.17) and (1.18) is called the **constrained optimization** problem. If f and c are linear, then the problem is called a **linear programming** problem. If all the constraints are linear and the objective function is quadratic, then the problem is a **quadratic programming** problem. If any of the functions is nonlinear, then we have a **nonlinear programming** problem [17, p. 257, p. 289].

The lowest of all minima is called the global minimum. Other minima are local. If the problem does not have special properties, there is no way to guarantee that the minimum value of a function found from conventional numerical method is the global one. Nevertheless, one idea is to start from different initial gausses, which are scattered through the feasible region. If they produce the same result, then there is a good chance that the global minimum has been found. If they produce different results, taking their minimum would be the possible desired global minimum. But there may be some unexplored regions with smaller values of the objective function [17, p. 258].

Definition 1.2 *[31]*

1. *A set*

$$N_f(x_0) := \{x \in Z : f(x) \leq f(x_0)\} \tag{1.20}$$

*is called the **level set** of (1.17-1.18) at x_0.*

2. *If $\phi : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function and $x, s \in \mathbb{R}^n$, then the **directional derivative** of ϕ is given by $\nabla\phi(x)^T s$ which is the rate at which ϕ changes at the point x in the direction s.*

3. *A vector $s \in \mathbb{R}^n$ is called the **descent direction** of a continuously differentiable function $\phi : \mathbb{R}^n \to \mathbb{R}$ at $x \in \mathbb{R}^n$ if the directional derivative $\nabla\phi(x)^T s < 0$.*

Definition 1.3 *[17]*

1. *A set $Z \in \mathbb{R}^n$ is **convex** if it contains the line segment between any two of its points $x, y \in Z$ i.e.,*

$$(1-t)x + ty \in Z \qquad \forall t \in [0,1]. \tag{1.21}$$

2. *A function $f : Z \to \mathbb{R}^n$ is **convex** on a convex set Z if its graph along any line segment lies on or below the chord connecting the functional values at the end points $x, y \in Z$ of the segment, i.e.,*

$$f((1-t)x + ty) \leq (1-t)f(x) + tf(y) \qquad \forall t \in [0,1] \tag{1.22}$$

*and the function is called **strictly convex** if*

$$f((1-t)x + ty) < (1-t)f(x) + tf(y) \qquad \forall t \in (0,1). \tag{1.23}$$

Convexity plays an important role in optimization. A strictly convex function has only one minimum. So its local minimum is also the global minimum. The term **local convergence** is used when a method converges to a local minimum, only if the initial point lies close enough to the solution. The term **global convergence** is used when a method converges to a local minimum, no matter where the initial point may lie. Newton's method with local convergence property will be called the **local Newton method**, or simply Newton's method. Newton's method with global convergence property will be referred to as the **global Newton method**.

Chapter 2

Fundamentals of numerical computing

2.1 Introduction

Numerical computing relies heavily on how numbers are represented and manipulated on a computer. To work on difficult problems with confidence, we need to have a sound understanding of the fundamentals of computer arithmetic. We need to know how number systems are designed and how arithmetic works. Humans use base 10 (decimal) arithmetic system for representing a number. Most computers express numbers in base 2 (binary) arithmetic system. The reason for changing from decimal to binary arithmetic is merely that it is very easy to do addition, subtraction, multiplication and division in binary arithmetic. However two problems arise in changing from decimal to binary:

1. we need to constantly convert numbers written in decimal by human to binary number system and then back again to decimal for humans to read the results.

2. we need to understand the limits of arithmetic in two different number systems.

A number can be represented in many ways. For example, a number 129.34 can be represented as 1.2934×10^2, 0.12934×10^3 or 1293.4×10^{-1}. To avoid such and other difficulties, the **Institute for Electrical and Electronics Engineers**(IEEE) fixed a standard [5, 24, 32] for representing floating-point numbers that are followed by almost every modern computer. In this chapter, we will analyse the number system and its limitation. We will also elaborate how to achieve the high performance in numerical computing. To illustrate these, we presented some C programs in this chapter. The programs were compiled by a GCC compiler of version 3.3.1 and run under a Linux Pentium 4 PC.

2.2 Integer representation

In the binary notation we have only two possible states, 0 and 1 which are known as **bits** (abbreviation for binary digits). We will use b_i to denote a bit at the ith position of a binary number. An integer is a number with no fractional part. It can be positive,

negative or zero. An **unsigned integer** is an integer that is equal to or greater than zero. A **signed integer** is an integer which is positive, negative or zero. An $(n+1)$ bit integer has the following form

$$b_0 b_1 \dots b_{n-1} b_n. \tag{2.1}$$

The value of an unsigned integer is given by

$$I_u = \sum_{i=0}^{n} b_i 2^{n-i}. \tag{2.2}$$

The range of values for an $(n+1)$ bit unsigned integer is from 0 to $2^{n+1}-1$. For an example, an integer number 10 can be stored in 32 bits as

00000000	00000000	00000000	00001010
1			32

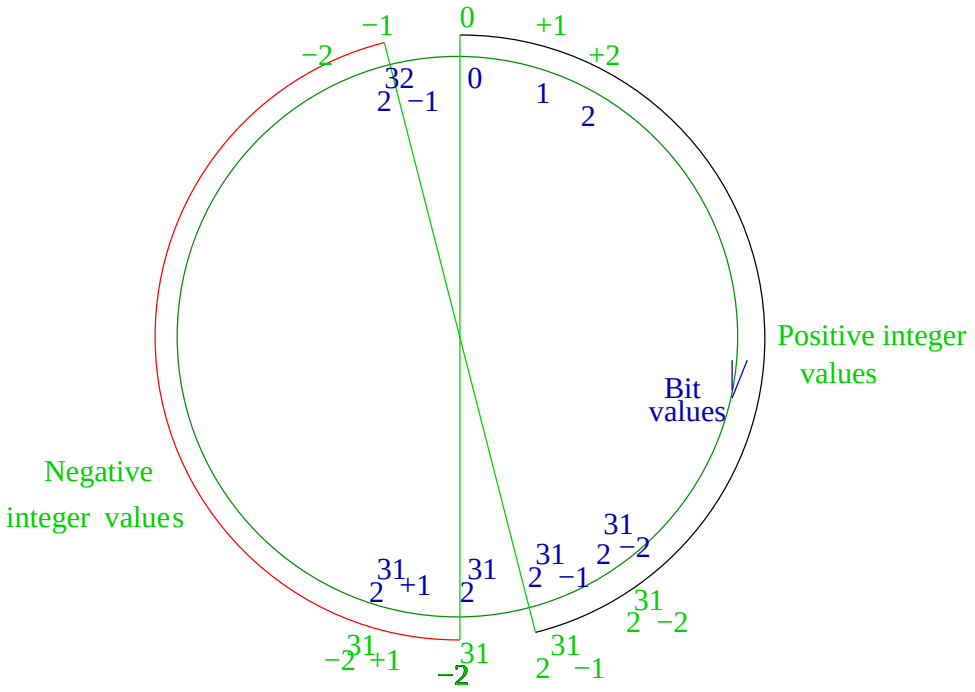

FIGURE 2.1 Illustration of integer value evaluation using 2's complement

Signed integers are represented in several ways including **sign-magnitude, 1's complement, 2's complement and biased representation**. Here we restrict our discussion to the 2's complement method which is used in most computer systems. In the 2's complement system, the value of an $(n+1)$ bit integer can be obtained from $b_0, b_1, \cdots, b_n$ by

$$I = \sum_{i=1}^{n} b_i 2^{n-i} - b_0 2^n. \tag{2.3}$$

The range of values for an $(n+1)$ bit integer I is between -2^n and $2^n - 1$. A 32 bits nonnegative I lies in the range between 0 and $2^{32}-1$ keeping the leading bit b_0 as zero

and storing the value in remaining 31 bits. A negative integer $I = -y$, where $1 \leq y \leq 2^{31}$, is stored as

$$2^{32} - y \tag{2.4}$$

in a binary form. FIGURE 2.1 shows how the integer value can be evaluated from a 32 bits format. The inner circle of the figure shows the bit values which increase from 0 to $2^{32} - 1$. The outer circle shows the associated integer values. The right side of this circle generates positive integers, while the left side yields negative integers which can be calculated using (2.3) from bits values of the inner circle. The binary format of a negative integer always contains 1 at its leading bit. Thus the leading bit distinguishes between positive and negative integers.

The **bit inversion** is the change of the state of a bit to the opposite state, i.e. the changing of a state 0 to 1 or a state 1 to 0. The 2's complement representation of a negative integer $-x$ from a positive integer x can be obtained by a bit inversion and an addition of 1 as demonstrated below for -8.

00000000 00000000 00000000 00001000	$= 8$
$\Downarrow$ Bit inversion	
11111111 11111111 11111111 11110111	$= 2^{32} - 1 - 8$
$\Downarrow +1$	
11111111 11111111 11111111 11111000	$= 2^{32} - 1 - 8 + 1 = -8$

Let us assume x and y are two integers. If we subtract y from x, this makes 0 when $x = y$. It can also be demonstrated for the 2's complement system as follows.

$$x - y = x + (-y) \tag{2.5}$$

The number $x + (-y)$ in 2's complement system is given by

$$\begin{aligned} & x + 2^{32} - y && (2.6)\\ = \; & x - y + 2^{32} && (2.7)\\ = \; & 0 + 2^{32} && (2.8) \end{aligned}$$

The number 2^{32} is too big to store for 32 bits, since it is comprised of one followed by 32 zeros

1	00000000	00000000	00000000	00000000

.

The leading bit 1 is abandoned, and that gives exactly 0 which we want. If $x > y$ in (2.5), (2.7) generates a value which again cannot fit in 32 bits. The leading bit 1 is discarded which is the same as omitting 2^{32} from (2.7). Hence, we get the exact value of $x - y$. If $x < y$, the result is a negative value. (2.6) can be rewritten as $2^{32} - (y - x)$. In this case the value fits in 32 bits. Comparing with (2.4) we observe that it is a negative value of

$y - x$ which we have anticipated [22, 32].

The addition of signed numbers in the 2's complement system is performed in a simple way. Two numbers -8 and 17 are added using 32 bits as follows.

00000000	00000000	00000000	00010001	17
+11111111	11111111	11111111	11111000	-8
00000000	00000000	00000000	00001001	9

The leading bit resulted by the above addition is omitted due to the reason mentioned before.

Overflow occurs when an integer is outside of the bits range. For example, the result of an addition $a+b$ of two integers a, b might not be representable in their range, although both a and b are representable. The following C program gives some unexpected result due to overflow.

Program 2.1 ***Overflow of integer numbers***

```
int main(){
    int a, b, c, d, x, y;
    a = 2147483647;
    b = 10;
    c = -2147483648;
    d = 9;
    x = a+b;
    y = c-d;
    printf("The value of x is %i\n", x);
    printf("The value of y is %i\n", y);
    return 0;
}
```

We get the following output:

```
The value of x is -2147483639
The value of y is  2147483639
```

Results from the above program are completely wrong due to overflow of integer numbers. We can explain this result using FIGURE 2.1.

$$\begin{aligned} x = a + b &= 2147483647 + 10 = 2^{31} + (-1 + 10) \\ &> 2^{31} - 1, \end{aligned}$$

which is outside the range of nonnegative signed integers. This can be written in the 2's complement system as

$$-2^{31} + 9 = -2147483639.$$

Similarly,

$$c - d \quad = \quad -2147483648 - 9 = -2^{31} - 9 < -2^{31},$$

which gives

$$2^{31} - 9 = 2147483639.$$

TABLE 2.1 shows ranges of different integer data types which are used by C/C++.

TABLE 2.1
Ranges of different integer data types [8]

Integer data type	Bits	Lower bound	Upper bound
short int	16	-32768	32,767
unsigned short	16	0	65,535
int	32	-2,147,483,648	2,147,483,647
unsigned int	32	0	4,294,967,295
long	64	-2^{63}	$2^{63} - 1$
unsigned long	64	0	$2^{64} - 1$

2.3 Floating-point representation

A floating-point number is represented by a triple, namely sign, exponent and fraction as follows.

$$V = \pm M \times \beta^E, \tag{2.9}$$

where M = **mantissa** or **significand**, β= **radix** or **base**, and E = **exponent** [18].

We restrict our attention to the binary system where $\beta = 2$, $0 \leq M < 2$. A floating-point number $V \neq 0$ is transformed into $\pm M \times \beta^E$ by changing the exponent E such that M has a leading nonzero bit. This transformation provides uniqueness of the number and is called **normalization**. The form obtained is called normalized representation. However, normalization cannot be applied to the case of $V = 0$. For representing zero, all bits of M must be zeros.

In the next sections, we will discuss the floating-point representation by the IEEE standard in details. We will use the notation $a\ e \pm b$ to express the number $a \times 10^{\pm b}$. A **number** is to be understood as a decimal number unless otherwise specified.

2.3.1 Single format

The IEEE single format to represent a floating-point number requires a 32 bits, which can be numbered from 0 to 31, left to right. The leading bit is the **sign bit** S, the next eight bits are the **characteristic bits** C and the final 23 bits are the fractional part F of the significand M. The leading bit of M is not stored as it is known that it has the value 1 for a normalized number.

Signed bit	Characteristic bits	Fractional bits
1-bit S	8-bit C	23-bit F
0	8	31

The sign bit S is 0 for positive numbers and 1 for negative numbers. The exponent E (from (2.9)) which is a signed integer is converted to C by adding a **bias** value, i.e.,

$$bias = \frac{1}{2}\beta^N \tag{2.10}$$

$$C = E + bias, \tag{2.11}$$

where N is the number of characteristic bits. So the bias for the single format is $\frac{1}{2} \times 2^8 = 127$. The exponent E lies in the range between -126 and 127. Therefore, the range of C is between 0 and 255. The idea of bias is to convert E into a nonnegative integer which can stored as binary bits. A floating-point number is evaluated by the IEEE single format by

$$V = (-1)^S 2^{C-127} F. \tag{2.12}$$

Now we turn to some examples with the single format. The number represented by the sequence of bits

1	10000111	00101100000000000000000

is negative because the sign bit is 1. Characteristic bits $C = 10000111$ give the number 135. Therefore, $E = C - 127 = 8$. Its mantissa is expressed by the binary bits 1.00101100000000000000000 where the initial 1 is the hidden bit and the remaining digits are taken from the last 23 bits of the above binary number. This part expresses the number

$$2^0 + 2^{-3} + 2^{-5} + 2^{-6} = \frac{75}{64}.$$

So the complete number is $-\frac{75}{64} \times 2^8$, which is -300.0.

The binary number

0	10000001	10100000000000000000000

with the leading bit, zero implies that the number is positive. Similar to the previous example, it can be evaluated as $+1 \times 2^{129-127} \times 1.101 = 6.5$.

Conversely, let us find the single format representation of a floating-point number 5.75. The sign bit is 0, since the number is positive. We express 5.75 in terms of a power of two and a mantissa M with $1 \leq M < 2$:

$$\begin{aligned} 5.75 = 23/4 &= (23/16) \times 2^2 = (1 + 4/16 + 2/16 + 1/16) \times 2^2 \\ &= (1 + 2^{-2} + 2^{-3} + 2^{-4}) \times 2^2 = (1.0111)_2 \times 2^2. \end{aligned}$$

The exponent E is 2. It is stored with a bias 127, and hence $C = 127 + 2 = 129 = (10000001)_2$. We line up the sign bit, the characteristic bits, and the bits following the

floating-point in the mantissa and we have

0	10000001	01110000000000000000000

.

Recall that the exponents permitted in single format numbers range between -126 and +127, and thus characteristic bits range between 00000001 and 11111110. The largest number N_{max} in the single format is given by

0	11111110	11111111111111111111111

,

where $C = 254, M = 2^{23} - 1$. It follows that

$$\begin{aligned} N_{max} &= V = +1 \times 2^{C-127} \times M \\ &= 340282346638528859811704183484516925440.0 \\ &= 3.40282347e + 38. \end{aligned}$$

When the characteristic bits are all ones, the number represented is not a floating-point number at all, but a conventional signal of a computation that has gone wrong, either by going above the largest representable floating-point or below the smallest, or by attempting some undefined arithmetic operation, such as dividing by zero or taking the logarithm of a negative number. We point out following exceptions that result from the number with all ones in characteristic bits.

The number

0	11111111	00000000000000000000000

with $C = 255$, $F = 0$, and $S = 0$ represents **positive infinity**, a pseudo-number that indicates that some unrepresentably large quantity has been generated by an arithmetic operation. Changing the sign bit to 1, the number

1	11111111	00000000000000000000000

yields **negative infinity**, an indication of a similar problem at the other end of the range. If some bits of the mantissa are ones, the pseudo-number represents NaN (**Not a Number**). The number

0	11111111	00000100000000000000000

with $C = 255$ and $F \neq 0$ gives NaN. Similarly with S = 1, the number

1	11111111	00000100000000000000000

generates NaN too. Positive infinity, negative infinity and NaN are not floating-point numbers. The appearance of a pseudo-number is supposed to be a danger signal to the programmer and usually results from a computational error. TABLE 2.3 shows different operations that produce these exceptions. Note that any operation with NaN will generate NaN too.

The smallest positive normalized floating-point number N_{min} in the single format is given by

0	00000001	00000000000000000000000

with $C = 1$, $F = 0$, and $S = 0$. Hence,

$$\begin{aligned} N_{min} &= V = +1 \times 2^{1-127} \times 1.0 = 2^{-126} \\ &= 1.17549435e - 38. \end{aligned}$$

We have not yet used any of the bit patterns in which the characteristic bits are all zeros.

A positive number below 2^{-126} is represented by a **subnormal**. In a subnormal number, all characteristic bits are zeros and the significand M has a zero at the leading bit position. For example, 2^{-127} in the binary form

0	00000000	10000000000000000000000

gives the number

$$V = +1 \times 2^{-126} \times 0.1 = 2^{-127} = 1.17549421e - 38.$$

Subnormal numbers are separable from normal numbers as all the characteristic bits C of a subnormal number are always zeros. The smallest positive binary number with this system is

0	00000000	00000000000000000000001

which gives

$$\begin{aligned} V &= +1 \times 2^{-126} \times 0.00000000000000000000001 = 2^{-149} \\ &= 1.40129846e - 45. \end{aligned}$$

Let us consider the following C program.

Program 2.2 ***Smallest number in the single format***

```
int main(){
    float x, y = 1;
    while (y > 0)
    {
        x = y;
        y = y/2;
    }
    printf("The value of x is  %f\n", x);
    printf("The value of y is  %f\n", y);
    return 0;
}
```

We get the following output

```
The value of x is  1.40129846e-45
The value of y is  0.0
```

Program 2.2 computes the smallest representable number x with the single format. The representable value less than x is 0. Zero can be represented in two different ways. In one, with $C = 0, F = 0$ and $S = 0$, we have

0	00000000	00000000000000000000000

which gives $V = +0$. The other one with $C = 0, F = 0$ and $S = 1$ gives

1	00000000	00000000000000000000000

.

Hence, $V = -0$. TABLE 2.2 gives a summary of preceding examples. Subnormal numbers are stored less accurately than normalized numbers since there are fewer significant digits in the mantissa, but without this special convention for characteristic bits with all zeros, it would not be possible to represent them at all. We will expand on this in Section 2.6.

TABLE 2.2
Bit values in the single format and their IEEE values [5]

Number	S	C	M	Decimal value
+0	0	0	0	+0.0
-0	1	0	0	-0.0
maximum normal number N_{max}	0	254	$2 - 2^{-23}$	3.40282347e+38
minimum positive normal number N_{min}	0	0	2^{-126}	1.17549435e-38
maximum subnormal number	0	0	2^{-127}	1.17549421e-38
minimum positive subnormal number	0	0	2^{-149}	1.40129846e-45
+Infinity	0	255	0	$+\infty$
-Infinity	1	255	0	$-\infty$
not a number	0 or 1	255	$F \neq 0$	NaN

TABLE 2.3
Operations that produce exceptions [5]

Operation	Set result to
$x/(\pm\infty)$	0
$\infty + \infty$	∞
$\infty + (-\infty)$	NaN
$(\pm\infty)/(\pm\infty)$	NaN
$0 \times \infty$	NaN
$0/0$	NaN
$\sqrt{x}$ when $x < 0$	NaN
$x/0$ when $x \neq 0$	$\pm\infty$

2.3.2 Double format

The IEEE double format is quite similar to the single format. A double format which is comprised of 64 bits begins with a sign bit. The next 11 bits are used for the characteristic which is a biased exponent (see equation (2.10) and (2.11)). The bias value is 1024 and the range of exponent for normalized numbers is from -1022 to +1023.

1-bit S	11-bit C	52-bit F
0	11	63

The remaining 52 bits are used for the mantissa. Like the single format representation, only the digits following the floating-point of the mantissa are actually stored. With double format the largest representable number is $2^{1023}(2-2^{-52})$, and the smallest representable positive number is 2^{-1074}. TABLE 2.4 shows ranges of double format numbers.

TABLE 2.4
Bit values in the double format and their IEEE values [5]

Number	S	C	M	Decimal value
+0	0	0	0	+0.0
-0	1	0	0	-0.0
maximum normal number	0	2047	$2-2^{-52}$	1.7976931348623157e+308
minimum positive normal number	0	0	2^{-1022}	2.2250738585072014e-308
maximum subnormal number	0	0	2^{-1023}	2.2250738585072009e-308
minimum positive subnormal number	0	0	2^{-1074}	4.9406564584124654e-324
+Infinity	0	2048	0	$+\infty$
-Infinity	1	2048	0	$-\infty$
not a number	0 or 1	2048	$F \neq 0$	NaN

2.4 Rounding error

The rounding error [32] is one characteristic feature of floating-point computation. Rounding errors occur due to the fact that the infinite number system of mathematics is transformed into a finite number system on the computer. Most computations with floating-point numbers produce values that cannot be represented exactly using specified bits. Therefore, the result of the floating-point computation is most often rounded to fit into its finite precision representation.

For example, let us examine the following C program.

Program 2.3 ***Rounding errors***

```
int main(){
    float y, z ;
    y = 2.52313e+03 ;
    z = 1.4 ;
    printf("y = \n  %12.11e\n", y);
```

```
    printf("z = \n  %12.11f\n", z);
    return 0;
}
```

The output from this program is

```
y =  2.52312988281e+03
z =  1.39999997616e+00
```

The difference between the value 2.52313×10^3 assigned to y and the value printed out is $0.00000011719 \times 10^3$ and the accuracy of representing y in the single format is about 6 to 7 significant digits. In other words, y has about 6 to 7 significant digits if it is to be represented in the single format. Similarly, the difference between the value 1.4 assigned to z and the value printed out is 0.00000002384, which is eight decimal orders of magnitude smaller than z. Therefore, the accuracy of representing z is about 7 to 8 significant digits.

This means that approximations are used for many floating-point numbers that cannot be exactly represented in binary bits. For instance, 7/5 is exactly 1.4 in decimal system, but 7/5 cannot be exactly expressed as a sum of powers of two. 7/5 has an infinite binary expansion

$$\boxed{1.\quad 01100110\quad 01100110\quad 01100110\quad 01100110\cdots}$$

In the single format, the expansion is rounded off at the twenty-third bit after the binary point. Thus 7/5 is not actually stored as 7/5, but as its close approximation

$$\begin{aligned} 1 + 2^{-2} + 2^{-3} + 2^{-6} + 2^{-7} + 2^{-10} + 2^{-11} + \cdots + 2^{-22} + 2^{-23} \\ = 11744051/8388608 = 1.399999976158. \end{aligned}$$

TABLE 2.5
Precision and machine epsilon of storage formats [32]

Format	Precision (binary significant digit)	Decimal significant digit	Machine epsilon
single	24	6-9	$\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$
double	53	15-17	$\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$
double extended	64	18-21	$\epsilon = 2^{-63} \approx 1.1 \times 10^{-19}$

The **precision** of a floating-point number is the number of bits in the significand. A floating-point number can be expressed as

$$\pm(b_0.b_1b_2\cdots b_{p-1}) \times 2^E, \tag{2.13}$$

where p is the **precision** of the floating-point system. For a normalized number, $b_0 = 1$ and for a subnormal number, $b_0 = 0$. The precision p of the single format is 24, often

called the single precision. Similarly, the double precision $p = 53$ implies the precision of the double format. The single precision corresponds to approximately

$$Log_{10}2^{24} = 7$$

significant decimal digits. Likewise, the double precision corresponds to approximately 16 significant decimal digits. TABLE 2.5 shows precisions and machine epsilons of different IEEE formats.

The **machine epsilon** ϵ is the gap between 1 and the next larger number. The number greater than 1 is $(1.00\ldots001)_2 = 1 + 2^{-(p-1)}$. So the value of ϵ is

$$\epsilon = 2^{-(p-1)}. \tag{2.14}$$

$Ulp(x)$ is the shorthand for **unit in the last place** which is actually the gap between a floating-point number x and its adjacent number with larger absolute value. Hence,

$$\begin{aligned} Ulp(x) &= (0.00\ldots001)_2 \times 2^E = 2^{-(p-1)} \times 2^E && (2.15)\\ &= \epsilon \times 2^E. && (2.16) \end{aligned}$$

TABLE 2.6 shows Ulp of different numbers with the single format.

TABLE 2.6
Ulp of different numbers with the single format [4]

Number (x)	Next representable value(x_+)	Ulp(x)
0.0	1.4012985e-45	1.4012985e-45
1.1754944e-38	1.1754945e-38	1.4012985e-45
1.0	1.0000001	1.1920929e-07
2.0	2.0000002	2.3841858e-07
16.000000	16.000002	1.9073486e-06
128.00000	128.00002	1.5258789e-05
1.0000000e+20	1.0000001e+20	8.7960930e+12
9.9999997e+37	1.0000001e+38	1.0141205e+31

The IEEE standard defines the **correctly rounded value** of x which we will denote by ***fl(x)***. Let x_- be the value that is obtained by truncation of the binary expansion of x after the bit length of a specific format. x_+ denotes the next higher value represented in that specified format. The correctly rounded value depends on which of the following round modes is in effect.

Round down: Rounding is performed towards $-\infty$

$fl(x) = x_-$

Round up: Rounding is performed towards ∞

$fl(x) = x_+$

Round to zero: Rounding is performed towards 0

$$fl(x) = x_- \text{ if } x \leq 0$$
$$fl(x) = x_+ \text{ if } x \geq 0$$

Round to nearest: Rounding is performed towards either x_- or x_+ which one is nearer. An exception is

$$\text{if } x > N_{max}, \text{ then } fl(x) = \infty,$$
$$\text{if } x < -N_{max}, \text{ then } fl(x) = -\infty,$$

where N_{max} is the largest representable floating-point number. If x_- and x_+ have the same distance from x, then the number which has 0 bit at its last bit is selected. To illustrate the round to the nearest for other cases, we extend the bit sequence of (2.13) for a precision p as follows.

$$\pm(b_0.b_1b_2\cdots b_{p-1}b_pb_{p+1}b_{p+2}\cdots) \times 2^E. \tag{2.17}$$

Note that b_{p-1} is the last storable bit and the bits $b_p,\ b_{p+1}, \cdots$ cannot be stored. If the bit b_p is 0, then round to nearest is x_-, and if $b_p = 1$ and at least one of the subsequent bits $b_{p+1}\cdots$ is nonzero, round to nearest is x_+. If $b_p = 1$, and all other subsequent bits are 0, there is a tie. The last bits of x_- and x_+ must be different. In case of a tie, the chosen number is which one of x_- and x_+ has 0 bit at the last bit position. According to the IEEE standard, the default round mode is round to nearest.

The **relative rounding error** corresponding to a nonzero number x is defined by

$$relerr(x) \quad = \quad \left|\frac{fl(x) - x}{x}\right|. \tag{2.18}$$

Let δ be such that

$$fl(x) = x(1 + \delta), \tag{2.19}$$

which gives

$$\delta = \frac{fl(x) - x}{x} \tag{2.20}$$

Therefore, using this result in (2.18) we find that

$$relerr(x) = |\delta|. \tag{2.21}$$

For any rounding mode we have

$$\begin{aligned} |fl(x) - x| \quad &< \quad (\text{ gap between } x_- \text{ and } x_+) \\ &= \quad Ulp(x) = 2^{-(p-1)}2^E, \end{aligned} \tag{2.22}$$

using (2.15).

We know from (2.13) that $x \geq 2^E$ for x in the normalized range. Using the above inequality, the relative rounding error of a normalized number must satisfy the bound

$$relerr(x) = \left|\frac{fl(x)-x}{x}\right| < \frac{2^{-(p-1)}2^E}{2^E} = 2^{-(p-1)} = \epsilon \tag{2.23}$$

for all rounding modes. In case of round to nearest we find that

$$\begin{aligned} |fl(x)-x| &\leq \frac{1}{2}(\text{ gap between } x_- \text{ and } x_+) \\ &= \frac{1}{2} \times 2^{-(p-1)} \times 2^E = \frac{1}{2}2^{-p} \times 2^E \end{aligned} \tag{2.24}$$

$$relerr(x) = \left|\frac{fl(x)-x}{x}\right| \leq \frac{2^{-(p)} \times 2^E}{2^E} = 2^{-(p)} = \frac{1}{2}\epsilon. \tag{2.25}$$

Let x and y be two floating-point numbers. The basic arithmetic operation can be given by

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \tag{2.26}$$

where op = +, -, * / and $|\delta| \leq \frac{1}{2}\epsilon$. The model implies that the computed value from an arithmetic operation is the same as the rounded value from the exact operation. For example, $x = 1$, and $y = 2^{-25}$. The exact sum of two numbers is

$$x + y = 1.\ 00000000\ 00000000\ 00000000\ 1,$$

which does not fit in the 23 fractional bits of the single format. The rounded result will be

$$x + y = 1.$$

Guard bits are the extra bits $b_p, b_{p+1}, \cdots$ beyond the last storable bit b_{p-1} of (2.17). Guard bits help to round the floating-point operation correctly. We need three extra bits b_p, b_{p+1}, b_{p+2} on the right of b_{p-1} to round the floating-point addition and subtraction correctly. The first two extra bits are called guard bits and the last extra bit is called the **sticky bit**. In an arithmetic operation we may need to shift the bits to the right by changing the exponent bits E accordingly. We call this process as **alignment**. If there is any nonzero extra bit discarded beyond the second guard bit during alignment, then the sticky bit is taken as 1. In other cases, the sticky bit is 0 [32].

Assume the following single format arithmetic operation of $x - y$ with $x = 1.0$ and $y = (1.00001)_2 \times 2^{-25}$.

$$
\begin{array}{cccccc}
(1.000 & 0000 & 0000 & 0000 & 0000 & 0000)_2 \times 2^0 \\
\text{-}(1.000 & 0100 & 0000 & 0000 & 0000 & 0000)_2 \times 2^{-25} \\
\hline
\multicolumn{6}{c}{\Downarrow \text{Alignment}} \\
(1.000 & 0000 & 0000 & 0000 & 0000 & 0000\ 000)_2 \times 2^0 \\
\text{-}(0.000 & 0000 & 0000 & 0000 & 0000 & 0000\ 011)_2 \times 2^0 \\
\hline
(0.111 & 1111 & 1111 & 1111 & 1111 & 1111\ 101)_2 \times 2^0
\end{array}
$$

Note that the bits of y are shifted to the right when alignment is performed. In the shifted bits we have two guard bits 0 and 1, and a sticky bit 1 (The bit at this position is 0 but beyond that we have a nonzero bit). After normalizing the result we have

$$(1.111 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \quad 1111\ 01)_2 \times 2^{-1}.$$

Finally, round to nearest gives

$$(1.111 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \quad 1111\ )_2 \times 2^{-1}.$$

Another dilemma with the floating-point arithmetic is the accumulation of rounding errors. While many applications run well in spite of a large number of calculations, some calculations encounter surprising problems. An illustration is, to add up many ones

$$n = 1 + 1 + 1 + \cdots .$$

The partial sums stop increasing at some point when the ratio $1/n$, that is of 1 to the current sum n, is smaller than the machine epsilon and the computation of this infinite series converges. At the point of convergence, we have

$$fl(n+1) = fl(n) = n. \tag{2.27}$$

The relative error of $n + 1$ is

$$
\begin{array}{lll}
 & \left|\frac{fl(n+1)-(n+1)}{n+1}\right| & \leq \epsilon \\
\Rightarrow & \left|\frac{n-(n+1)}{n+1}\right| & \leq \epsilon \quad \{\text{using (2.27) }\} \\
\Rightarrow & \left|\frac{1}{n+1}\right| & \leq \epsilon.
\end{array}
$$

The minimum value of n is given by

$$
\begin{array}{lll}
 & \frac{1}{n+1} & \leq \epsilon \\
\Rightarrow & n+1 & \geq \dfrac{1}{\epsilon} \\
\Rightarrow & n & \geq \dfrac{1}{\epsilon} - 1 \approx \dfrac{1}{\epsilon}.
\end{array}
$$

So the infinite series of ones converges to $\frac{1}{\epsilon}$. Changing to the double format from the single format raises this limit of accuracy sufficiently far. But it does not completely eliminate the problem.

2.5 Overflow

Let us consider the following C program to compute the 2-norm of a vector.

Program 2.4 ***Overflow of floating-point numbers***

```
float Norm(float *a, int dim){
    int i;
    float s = 0;
    for(i = 0; i < dim; i++){
        s += a[i] * a[i];
    }
    s = sqrt(s);
    return s;
}

int main(){
    float a[3], x;
    a[0] = 1.5e20;
    a[1] = 2.2e20;
    a[2] = 1.9e20;
    x = Norm(a, 3);
    printf("The value of 2-norm is %g\n", x);
}
```

We get the following output.

```
The value of 2-norm is inf
```

However, we can determine the 2-norm as $3.271e20$ by hand computation. The output of the above program is infinity as the squares of array elements are larger than the largest representable number N_{max}. When a computed finite number is beyond the floating-point range $\pm N_{max}$ in the IEEE arithmetic, the value is set to be $\pm\infty$. As the value beyond that range cannot be represented, the state is termed as overflow. Since it is approximated to $\pm\infty$, overflow often brings futile computation. It is almost always worthless in matrix computations as it turns into NaN soon.

Specifically, in case of round to nearest, this is the same as saying that overflow occurs when an exact finite result is rounded to $\pm\infty$, but it is not the same for the other rounding modes. For example, in case of round down or round towards zero, if an exact finite result x is more than N_{max}, it is rounded down to N_{max} unless $x \geq N_{max} + Ulp(N_{max})$. Overflow occurs when $x \geq N_{max} + Ulp(N_{max})$ and the value is set to $+\infty$, since otherwise the rounded value would be the same even if the exponent range were increased [24].

2.6 Underflow

Now we consider the same function Norm() as in Program 2.4 with the following function main().

Program 2.5 ***Underflow of floating-point numbers***

```
int main(){
    float a[3], x;
    a[0] = 1.5e-23;
    a[1] = 2.2e-23;
    a[2] = 1.9e-23;
    x = Norm(a, 3);
    printf("The value of 2-norm is %g\n", x);
}
```

We get the following output.

```
The value of 2-norm is 0
```

But the result of this 2-norm from hand computation is $3.271e$-23. The output of the program is 0 as the squares of the array elements are smaller than the smallest subnormal number. This phenomenon is known as underflow.

Underflow occurs, roughly speaking, when the result of an arithmetic operation is so small that it cannot be stored in its intended normalized format. In the IEEE arithmetic, the standard response to underflow is to return the correctly rounded value, which is a subnormal number, between $-N_{min}$ and $+N_{min}$, where $+N_{min}$ is smallest positive normalized number. Subtracting two (positive) tiny numbers that are near the smallest normal number might produce a subnormal number. Or, dividing the smallest positive normal number by two produces a subnormal result. Producing subnormal numbers (rather than returning the answer zero) when the mathematically correct result has a magnitude less than the smallest positive normal number is known as gradual underflow. The presence of subnormal numbers provides greater precision to floating-point calculations that involve small numbers, although the subnormal numbers themselves have fewer bits of precision than normal numbers. Its use provides many valuable arithmetic rounding properties and significantly adds to the reliability of floating-point softwares. In absence of gradual underflow, user programs need to be sensitive to the implicit inaccuracy threshold since zero is used to replace a subnormal number. In case of no subnormal numbers, programmers need to implement their own method of detecting when they are approaching this inaccuracy threshold, or else abandon the quest for a robust, stable implementation of their algorithm [24].

On the other hand, the arithmetic with subnormal numbers is too complicated to justify the inclusion as a hardware operation which will be needed only occasionally. Even today, some IEEE compliant microprocessors support gradual underflow only in software.

The representable positive number smaller than the smallest positive subnormal is +0.0. Program 2.2 shows that we can reach to subnormal numbers. Now how can we eliminate overflow and underflow problems of the function Norm()? The idea is scaling. Algorithms can be reorganized by scaling the elements of the vector so that computations do not take place near zero or $\pm N_{max}$.

The following program reorganizes Norm() to avoid underflow and overflow effects. This is a modified C version of the Fortran program "snrm2.f" in BLAS(Basic Linear Algebra Subprograms) [3].

Program 2.6 ***Remedies of overflow and underflow***

```
float Norm(float *a, int dim){
    float value, sq, scale, absxi, temp;
    sq = 1;
    scale = fabsf(a[0]);
    for(i = 1; i < dim; i++){
        if (a[i] != 0){
            absxi =  fabsf(a[i]);
            if (scale < absxi){
                temp = scale/absxi;
                sq = 1 + sq * temp * temp;
                scale = absxi;
            }
            else {
                temp = absxi/scale;
                sq = sq + temp * temp;
            }
        }
    }
    value = scale * sqrt(sq);
return value;
}
```

We illustrate the structure of the program by an example vector, $a = [2, 5, -7, 1, 3]$

$$\begin{aligned} loop1 : sq &= 1 + \left(\frac{2}{5}\right)^2 \\ scale &= 5 \\ loop2 : sq &= 1 + \left(1 + \left(\frac{2}{5}\right)^2\right) * \left(\frac{5}{7}\right)^2 \\ scale &= 7 \\ loop3 : sq &= 1 + \left(1 + \left(\frac{2}{5}\right)^2\right) * \left(\frac{5}{7}\right)^2 + \left(\frac{1}{7}\right)^2 \end{aligned}$$

$$\begin{aligned} scale &= 7 \\ loop4 : sq &= 1 + \left(1 + \left(\frac{2}{5}\right)^2\right) * \left(\frac{5}{7}\right)^2 + \left(\frac{1}{7}\right)^2 + \left(\frac{3}{7}\right)^2 \\ scale &= 7 \\ value &= 7 * \sqrt{1 + \left(1 + \left(\frac{2}{5}\right)^2\right) * \left(\frac{5}{7}\right)^2 + \left(\frac{1}{7}\right)^2 + \left(\frac{3}{7}\right)^2} \end{aligned}$$

Now it is easy to verify that the calculated *value* is the 2-norm of the vector a, and overflow and underflow are avoided during computation. However, scaling the variables and detecting the inaccuracy threshold can be time-consuming for each numerical program.

2.7 Cancellation

Cancellation [18, 32] is a phenomenon which occurs when two nearly equal numbers are subtracted from each other resulting in a number with a loss of precision. The reason of the loss of precision is that leading digits of two numbers cancel. For example, assume $x = 100, y = 3$ and $u = 3, v = 100$. Now let us perform the following operations.

$$\begin{aligned} a &= \frac{x^2}{y}; \\ b &= a \times v; \\ c &= (a^2 - b); \end{aligned}$$

From above operations, we deduce that the exact value of c is 0. As arithmetic operations on decimal numbers are easy to conceptualize, we consider above operations in the decimal arithmetic. We will take a precision of 6 significant decimal digits (M has 6 digits in (2.9)). Then we deduce

$$\begin{aligned} a &= 100.0 \times 100.0/3.0 = 3333.33 = 3.33333 \times 10^3 \quad \text{(round to nearest)} \\ b &= 3.33333 \times 10^3 \times 3.0 = 9.99999 \times 10^3 \\ c &= (100.0 \times 100.0 - 9999.99) = 1.00000 \times 10^4 - 9.99999 \times 10^3. \end{aligned}$$

Aligning the numbers and using a guard bit for 9.99999×10^3 reveal that

$$\begin{aligned} c &= 1.000000 \times 10^4 - 0.999999 \times 10^4 \\ &= 0.01 = 10^{-2}. \end{aligned}$$

Here cancellation occurs when c is being computed. Instead of a small error in the sixth digit, we now have that error in the second digit. The absolute error in c has effectively been magnified by a factor 10^4 due to cancellation of leading 4 digits. The problem with cancellation is that the accuracy that a specified format aims to achieve is corrupted by the loss of leading significant digits. In many cases, cancellation may be prevented by rearranging the computation.

Consider the following arithmetic operation.

$$1 - (1 - 2t)^{-1} = \frac{1 - 2t - 1}{1 - 2t} = \frac{-2t}{1 - 2t}.$$

In this case, significant cancellation occurs when t is small. Cancellation becomes catastrophic when $|t| \leq \epsilon$. Using 6 significant decimal digits, for $t = 0.001$, the left hand side gives

$$1 - (1 - 2t)^{-1} = 1.00000 - 1.00200 = 2.00000 \times 10^{-3},$$

while the right hand side gives

$$\frac{-2t}{1 - 2t} = \frac{-2 \times 0.001}{1 - 2 \times 0.001} = 2.00400 \times 10^{-3}.$$

The relative error in using the left hand side is 0.002 which is often unacceptable. For $t = 10^{-7}$, the left hand side leads to a complete cancellation yielding zero and a relative error of one. However, we can circumvent cancellation using the right hand side.
Serious cancellation occurs in computing $1 - e^{-t}$ with 6 significant decimal digits when t is very small. We compute the values of e^{-t} by a calculator with the double extended format and consider 6 significant decimal digits. We find for $t = 0.0001$ that

$$1 - e^{-t} = 1 - 9.99900 \times 10^{-1} = 1.0 \times 10^{-5}.$$

Using its power series expansion we have that

$$1 - e^{-t} = 1 - \left(1 - t + \frac{t^2}{2} - \ldots\right) \approx t - t^2/2 = t(1 - t/2).$$

With $t = 0.0001$ the rightmost part leads to

$$t(1 - t/2) = 0.0001 \times (9.99950 \times 10^{-1}) = 9.99950 \times 10^{-5},$$

which properly approximates the result to six decimal digits. With $t = 10^{-5}$ and $t = 10^{-6}$, computations yield similar results. Complete cancellation occurs using original operation $1 - e^{-t}$ with $t = 10^{-7}$ while the approximation $t(1 - t/2)$ gives the correct result. Hence, sometimes an approximation will be more accurate than its original operation with the cancellation effect.

An enlightening illustration of cancellation is the numerical differentiation of a smooth function $f(x)$, where the derivative is approximated by the following *forward finite difference* scheme.

$$f'(x) \approx \frac{f(x + h) - f(x)}{h}.$$

Mathematically, the accuracy of this approximation is enhanced by taking a small h. We

can estimate the error by a Taylor series expansion as follows.

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \ldots \tag{2.29}$$

$$\Rightarrow f'(x) = \frac{f(x+h) - f(x)}{h} \underbrace{-\frac{1}{2}hf''(x) - \ldots.}_{O(h)} \tag{2.30}$$

Here the discretization error is the order of $O(h)$. Taking a large h will enlarge the discretization error in the above approximation. On the other hand, the functional values $f(x)$ and $f(x+h)$ can be computed only to a limited precision and for a small h and cancellation occurs in the subtraction $f(x+h) - f(x)$. Taking a smaller h yields a larger discrepancy due to more cancellation, which is magnified by dividing by a small h. We have to choose such an h that balances between the two errors. The forward difference scheme gives a good result with $h = \sqrt{\epsilon}$ [33]. A more accurate approximation of the numerical differentiation can be obtained using a *center difference* scheme which is defined by

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{2.31}$$

The discretization error in (2.31) can be estimated by the Taylor series expansion.

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3) \tag{2.32}$$

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3). \tag{2.33}$$

By subtracting (2.33) from (2.32), and dividing through $2h$ we obtain

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

Hence, the discretization error for center difference scheme is of the order $O(h^2)$ instead of $O(h)$. The center difference scheme gives good results with $h = \epsilon^{2/3}$ [33].

A common cancellation problem in statistics arises when we use the formula

$$\sum_{i=1}^{n} x_i^2 - n\overline{x}^2 = \sum_{i=1}^{n} x_i^2 - \overline{x}\sum_{i=1}^{n} x_i$$

for computing the sum of squares around the mean. Cancellation can be avoided by replacing the formula by

$$\sum_{i=1}^{n} (x_i - \overline{x})^2.$$

To illustrate the effect of cancellation, take a simple problem of $n = 5$ observations with $x_i = 3202 + i$ for $i = 1, 2, \cdots, 5$. Again using six decimal digits, the computations of the sum and mean encounter no problems, and we easily get $\overline{x} = 3205 = 3.20500 \times 10^3$ and

$\sum_i x_i = 16025 = 1.60250 \times 10^4$. However, each square loses some precision in rounding.

$$
\begin{aligned}
x_1 &= 3203x_1^2 = 10259209 \text{ rounded to } 1.02592 \times 10^7 \\
x_2 &= 3204x_2^2 = 10265616 \text{ rounded to } 1.02656 \times 10^7 \\
x_3 &= 3205x_3^2 = 10272025 \text{ rounded to } 1.02720 \times 10^7 \\
x_4 &= 3206x_4^2 = 10278436 \text{ rounded to } 1.02784 \times 10^7 \\
x_5 &= 3207x_5^2 = 10284849 \text{ rounded to } 1.02848 \times 10^7
\end{aligned}
$$

Summing the squares encounters no further rounding on its way to 5.13600×10^7 and we find that

$$
\begin{aligned}
\sum_{i=1}^{n} x_i^2 - \overline{x} \sum_{i=1}^{n} x_i &= 5.13600 \times 10^7 - 3.20500 \times 10^3 * 1.60250 \times 10^4 \\
&= 5.13600 \times 10^7 - 5.13601 \times 10^7 = -1.00000 \times 10^2.
\end{aligned}
$$

In the above operation of subtraction, cancellation takes place and the solution has only 1 significant digit after losing leading 5 digits. The other formula gives the perfect result which is 10 in this case.

While this example shows an absurd result, a negative value generated from the sum of squares due to rounding and cancellation errors, the second formula can avoid the computational difficulty and reduce the effect of limited precision arithmetic.

2.8 High performance computing

Operations involving matrices and vectors lie at the center of numerical computing. Therefore, it is essential to speed up such operations to achieve higher performance in numerical computing. These operations are typically expressed as sums over all matrix or vector indices, that is, (multiple) loops in a program. In an effort to speed-up and optimize the CPU time on a computer, one of the techniques used is **loop unrolling** [19].

Loop unrolling is an implementation technique where the number of loop iterations is reduced by calling multiple instructions in a single loop iteration. With the loop unrolling technique, a computer executes multiple instructions in a single loop iteration. The gain in performance from this technique can be described by **pipelining**.

Pipelining is a mechanism where multiple instructions are overlapped in execution. A computer pipeline consists of several stages. Each stage completes a part of an instruction in parallel. The stages form a pipe by connecting one stage to the next. The instructions enter at one end, progress through the stages, and exit at the other end. To keep a pipeline full, parallelism among instructions must be exploited by finding sequences of unrelated instructions that can be overlapped in the pipeline.

The following two programs compute $dy = da \times dx + dy$ with $dx \in \mathbb{R}^n$, $dy \in \mathbb{R}^n$ and $da \in \mathbb{R}$. This operation (constant $\times$ vector + vector) is called the AXPY operation.

Program 2.7 ***AXPY operation without loop unrolling***

```
void daxpy(int n, double da, double *dx, double *dy){
    for (i = 0; i < n ; ++i ){
        dy[i] = dy[i] + da * dx[i];
    }
    return;
}
```

As we notice in the above program, as a loop executes, only one floating-point operation is performed per loop iteration. However, in the next program, multiple floating-point operations are performed during a single loop iteration. Thus, on execution, AXPY operation with loop unrolling runs faster than that without loop unrolling for an enough large n. This is a modified version of the C program "daxpy.c" in CBLAS (C version of BLAS) [1].

Program 2.8 ***AXPY operation with loop unrolling***

```
void daxpy(int n, double da, double *dx, double *dy){
    int ix, iy, i, m;
    m = n % 4;
    if ( m != 0 ) {
        for ( i = 0; i < m ; i++ )
        dy[i] = dy[i] + da * dx[i];
        if ( n < 4 ) {return;}
    }
    for ( i = m; i < n ; i = i + 4 ) {
        dy[i] = dy[i] + da * dx[i];
        dy[i+1] = dy[i+1] + da * dx[i+1];
        dy[i+2] = dy[i+2] + da * dx[i+2];
        dy[i+3] = dy[i+3] + da * dx[i+3];
    }
    return;
}
```

There is a wide variety of storage in a computer system, which can be organized in a hierarchy (FIGURE 2.2) according to either their speed, cost or capacity. The higher levels are expensive, but very fast. As we move down the hierarchy, the cost decreases, while the access time increases and the amount of storage at each level increases. Data is normally stored in some storage system A (e.g., main memory). As it is used, it is copied into a faster storage system B (e.g., cache) on a temporary basis. When a particular piece of data is needed, the computer first checks if it is in B. If it is, the computer uses the data directly from B. If not, the computer uses the data from A, putting a copy in B in the hope that it will be needed again.

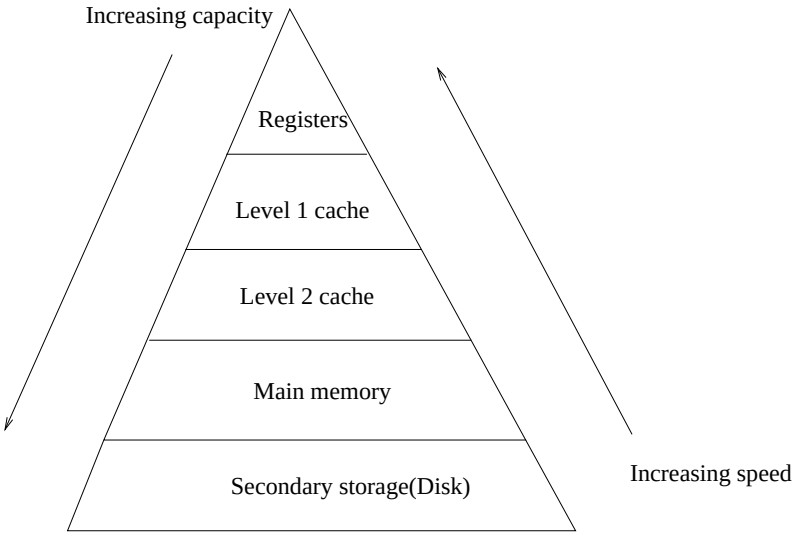

FIGURE 2.2 Illustration of memory hierarchy

In most computers data flows from memory to registers where given operations on the data are performed. Most modern CPUs are so fast that for most program workloads, slow memory transfer between different levels of the hierarchy is the practical limitation on processing speed. As a result, the CPU spends much of its time waiting for memory transfer to complete. Performance of the algorithms can be dominated by memory transfers, rather than the *flops* (floating point operations) involved. Higher performance can be achieved by minimizing the memory transfers and by reusing recently accessed items in the faster memories. FIGURE 2.2 suggests that we should try to hold the most frequently accessed items close to the CPU and successively larger (and slower, and less expensive) memories as we move away from the CPU.

Cache memories which are located between the CPU and main memories are typically five to ten times faster than main memories. The cache has an important role in computer systems. Since caches have limited size, cache management is an important design problem. Careful selection of cache size and a replacement policy can result in very high performance [14, 41].

We consider the following C programs of matrix-matrix multiplication $c = c + a \times b$ with $a \in \mathbb{R}^{m \times p}, b \in \mathbb{R}^{p \times n}$ and $c \in \mathbb{R}^{m \times n}$. We assume that the matrices are large and can be held in the main memory and the cache is large enough to hold two rows of a matrix but not a whole matrix.

Program 2.9 ***Matrix-matrix multiplication*** ijk ***version***

```
#include <stdio.h>

void unblockmatmul(double **a, double **b, double **c, int m, int n, int p)
{
   int i,j,k;
   for(i=0; i < m; i++){
```

```
      for(j=0; j < n; j++){
        for(k=0; k < p; k++){
          c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
      }
    }
}
```

Program 2.10 ***Matrix-matrix multiplication*** ikj ***version***

```
#include <stdio.h>

void unblockmatmul(double **a, double **b, double **c, int m, int n, int p)
{
   int i,j,k;
   for(i=0;i < m;i++){
     for(k=0;k < p;k++){
       for(j=0;j < n;j++){
         c[i][j]=c[i][j]+a[i][k]*b[k][j];
       }
     }
   }
}
```

The ikj version in Program 2.10 is faster than ijk version in Program 2.9. The reason is the difference in memory management. Note that elements of an array are stored row-wise in C language. We analyse Program 2.9 for the innermost loop k as follows.

1. $c[i][j]$ is a constant, and can therefore be stored in register;
2. The ith row of a is read which can be stored in cache;
3. b is read column-wise, and therefore it cannot be stored in the cache since only 2 rows can be stored in cache according to our assumption. Therefore, for each k, $b[k][j]$ is fetched from the main memory.

Program 2.10 is analysed for the innermost loop j as follows.

1. $a[i][k]$ can be stored in register.
2. The ith row of c can be stored in the cache;
3. The kth row of b can be stored in the cache;

We observe that Program 2.10 makes better use of cache memory which makes it faster [12, 35].

Now we present the following C program of *blocked matrix-matrix multiplication* with loop unrolling which is the modified version of blocked matrix-matrix multiplication in [42]. This program is much faster than Program 2.10.

Program 2.11 *Blocked matrix-matrix multiplication with loop unrolling*

```
#include <stdio.h>

#define min(a,b) ((a) < (b) ? (a) : (b))

/*  Note that the block size can be  optimized by trying with different values */

void blockmatmul(double **a, double **b, double**c, int m, int n, int p){
  int kb, ib, jb, i, j, k, blkk = 40;
  int ke, ie, je;
  double t1, t2, t3, t4;

  for (ib=0;ib < m;ib=ib+blkk){
    ie = min(ib+blkk-1,m-1);
    for (kb=0;kb < p;kb=kb+blkk){
      ke = min(kb+blkk-1,p-1);
      for(jb=0;jb < n;jb=jb+blkk){
        je = min(jb+blkk-1,n-1);
        for(i=ib;i <ie;i=i+2){
          for(j=jb;j <je;j=j+2){
            t1 = 0.0e0;
            t2 = 0.0e0;
            t3 = 0.0e0;
            t4 = 0.0e0;
            for(k=kb;k <=ke;k++){
              t1 = t1 + a[i][k  ]*b[k][j  ];
              t2 = t2 + a[i+1][k]*b[k][j  ];
              t3 = t3 + a[i][k  ]*b[k][j+1];
              t4 = t4 + a[i+1][k]*b[k][j+1];
            }
            c[i  ][j  ] = c[i  ][j  ]+t1;
            c[i+1][j  ] = c[i+1][j  ]+t2;
            c[i  ][j+1] = c[i  ][j+1]+t3;
            c[i+1][j+1] = c[i+1][j+1]+t4;
          }
        }
      }
    }
  }

  if (m-i == 1){
    for (jb = 0; jb < n; jb++){
```

```
      t1 = c[m-1][ jb];
      for (kb = 0; kb < p; kb++){
        t1 +=  a[m-1][kb] * b[kb][jb];
      }
      c[m-1][jb] = t1;
    }
  }

  if (n-j == 1){
    for(ib = 0; ib < i; ib++){
      t1 = c[ib][n-1];
      for(kb = 0; kb < p; kb++){
        t1 +=   a[ib][kb] * b[kb][n-1];
      }
      c[ib][n-1] = t1;
    }
  }
}
```

In the blocked multiplication, matrices are partitioned into blocks in such a way that the blocks can be stored in the cache memory. When operations with the cache data are accomplished, a new block of data overwrites in the cache memory. Thus the blocked multiplication works faster than simple multiplication by this cache management policy [12].

Performance of operations involving matrices and vectors can be evaluated by a number q which is the ratio of flops to memory transfers. For example, in the AXPY operation, we have one multiplication and one addition for n components of dx (dy) which make $2n$ floating point operations. We need to read n components of vectors dx, dy and 1 value of da from slow memory to register and write n components of dy back to slow memory. Therefore, the number of memory transfers is $3n + 1$.

The q value tells roughly how many flops we can perform per memory transfers or how many useful work is done with respect to the time of data moving. Since time of data moving between memories is much higher than the time of flops, larger q values give better performance in algorithms.

Simple operations like AXPY, scalar products, multiplying a vector by a scalar perform $O(n)$ flops and give the worst q values. Operations such as matrix-vector multiplications, solving triangular systems of equations perform $O(n^2)$ flops and yield slightly better q values. Operations such as matrix-matrix multiplications, solving triangular systems of equations with many right-hand sides perform $O(n^3)$ flops and offer the best q values. Therefore, we endeavor to rearrange algorithms in terms of operations such as matrix-matrix multiplications [12].

Many scientific and technical problems turn out to be in the form of $Mx = b$. Here M is a real $n \times n$ matrix and b and x are both real vectors of length n. M may be a

dense or sparse matrix. Here we will describe the Cholesky decomposition for factorizing a dense symmetric positive definite M which can later be used for solving $Mx = b$.

2.8.1 Chelosky decomposition

We present the Cholesky decomposition in the following algorithm.

Algorithm 2.12 ***Cholesky decomposition***

Given a symmetric $M \in \mathbb{R}^{n\times n}$,
for $k = 1, 2, \cdots, n$
  $tmp = 0$;
  for $i = 1, 2, \cdots, k-1$
    $tmp = tmp + L_{ki}^2$;
  end
  $L_{kk} = \sqrt{M_{kk} - tmp}$;
  if$(L_{kk} \leq 0)$*, then stop with false (Cholesky fails);*
  for $j = k+1, k+2, \cdots, n$
    $tmp = 0$;
    for $i = 1, 2, \cdots, k-1$
      $tmp = tmp + L_{ji}L_{ki}$;
    end
    $L_{jk} = \dfrac{M_{jk} - tmp}{L_{kk}}$;
  end
end
Stop with true (Cholesky succeeds);

Now we describe how we deduce Algorithm 2.12. Let $M \in \mathbb{R}^{n\times n}$ be symmetric positive definite. We can show that there exists a factorization $M = LL^T$, where L is a lower triangular matrix. This factorization is called Cholesky decomposition [29].
Entrywise we have

$$\begin{bmatrix} L_{11} & & \\ \vdots & \ddots & \\ L_{n1} & \cdots & L_{nn} \end{bmatrix} \begin{bmatrix} L_{11} & \cdots & L_{n1} \\ & \ddots & \vdots \\ & & L_{nn} \end{bmatrix} = \begin{bmatrix} M_{11} & \cdots & M_{1n} \\ \vdots & & \vdots \\ M_{n1} & \cdots & M_{nn} \end{bmatrix},$$

whence

$$\begin{aligned} j = k: \quad M_{kk} &= L_{k1}^2 + \ldots + L_{k,k-1}^2 + L_{kk}^2 \\ j > k: \quad M_{jk} &= L_{j1}L_{k1} + \ldots + L_{j,k-1}L_{k,k-1} + L_{jk}L_{kk}. \end{aligned}$$

The following formula gives the entries of L.

$$\begin{aligned} \text{Diagonal element, } L_{kk} &= \sqrt{M_{kk} - \sum_{i=1}^{k-1} L_{ki}^2} \\ \text{Other element, for } j > i : L_{jk} &= \left(M_{jk} - \sum_{i=1}^{k-1} L_{ji} L_{ki} \right) / L_{kk}. \end{aligned}$$

Algorithm 2.12 can be rearranged using the block concept. A crucial factor for performance is the choice of the block size b. Machine dependent parameters such as cache size and memory bandwidth will dictate the best choice for the block size.

The algorithm presented below is formulated from CLAPACK (C version of Linear Algebra PACKage) [2]. We will use Matlab notation $M(i : m, j : n)$ to denote the submatrix of M lying in rows i through m and columns j through n.

Algorithm 2.13 ***Blocked Cholesky decomposition***

for $k = 1 \cdots n$ *with step* b

$M(k : k+b, k : k+b) = M(k : k+b, k : k+b)$
$\qquad -L(k : k+b, 1 : k)L(k : k+b, 1 : k)^T;$

Use Algorithm 2.12 to factorize

$M(k : k+b, k : k+b) = L(k : k+b, k : k+b)L(k : k+b, k : k+b)^T$

$M(k+b : n, k : k+b) = M(k+b : n, k : k+b)$
$\qquad -L(k+b : n, k : k+b)L(k : k+b, 1 : k)^T;$

$L(k+b : n, k : k+b) = L(k : k+b, k : k+b)^{-1} M(k+b : n, k : k+b);$

end

The main advantage of the blocked algorithm of the Cholesky decomposition is that the operations involve a matrix-matrix operation, and therefore result in high performance. Similarly, we can use the blocking concept for other factorization methods like LU decompositions [12] and QR decompositions [12] as in CLAPACK [2].

Chapter 3

Nonlinear unconstrained optimization

3.1 Introduction

We consider the following unconstrained optimization problem:

$$\min_x f(x), \tag{3.1}$$

where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function.

The problem (3.1) may have different type of solutions. They are defined as follows.

Definition 3.1 *[31]*

1. ***Global Minimum**: A point x^* is a global minimum if $f(x^*) \leq f(x),\ \forall x$.*
2. ***Local minimum**: A point x^* is a local minimum if $f(x^*) \leq f(x),\ \forall x \in \mathbb{B}(x^*)$.*
3. ***Strict local minimum**: A point x^* is a strict local minimum if $f(x^*) < f(x)$ for all $x \in \mathbb{B}(x^*)$ with $x \neq x^*$.*
4. ***Isolated local minimum**: A point x^* is an isolated local minimum if x^* is the only minimum of f in some neighborhood of $\mathbb{B}(x^*)$.*

3.2 Condition of a minimum

FIGURE 3.1 shows the plot of a 1-dimensional function. We observe that the first derivative (gradient) of the function is zero at the maximum or minimum point which is a statement of Theorem 3.2 of the first-order necessary conditions. To distinguish between a minimum and a maximum we need to know the second derivative of the function which is positive for an isolated minimum. This is described as the second-order sufficient conditions in Theorem 3.4.

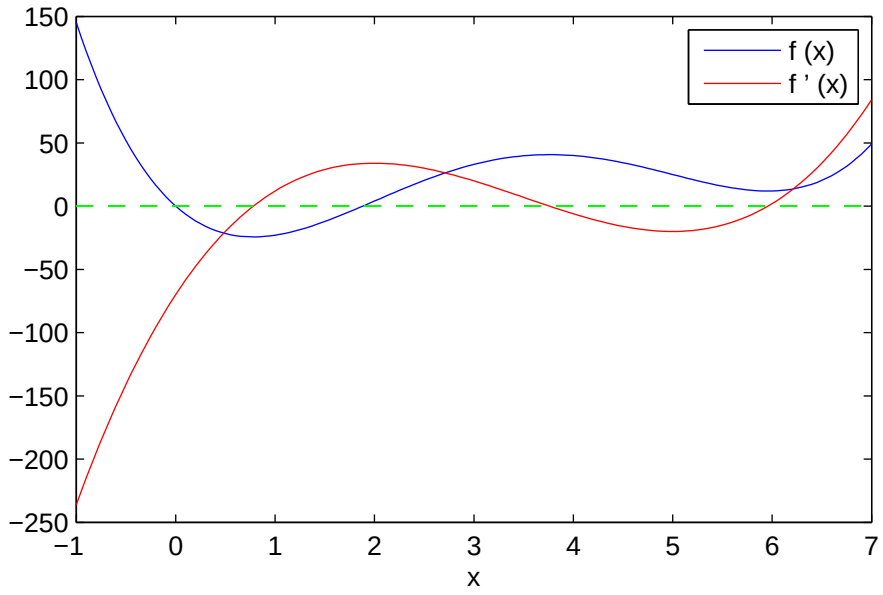

FIGURE 3.1 Plot of a function $f(x) = x^4 - 14x^3 + 60x^2 - 70x$ and its first derivative. The minimum or maximum lies at the point where the first derivative f' of the function is zero.

Theorem 3.2 ***First-order necessary conditions [40]***
If $x^* \in \mathbb{R}^n$ is a local minimum of f and f is continuously differentiable for all $x \in \mathbb{B}_\epsilon(x^*)$, then the gradient $g(x^*) = 0$.

Now we state the following definition.

Definition 3.3 *[40] Let $x^* \in \mathbb{R}^n$ and $f : U \to \mathbb{R}$ is differentiable in an open neighborhood U of x^*. If $g(x^*) = 0$, then x^* is called a stationary or critical point of f.*

Theorem 3.4 ***Second-order sufficient conditions [40]***
Suppose that f is twice continuous differentiable for all $x \in \mathbb{B}_\epsilon(x^*)$ and that

1. *the gradient vector $g(x^*) = 0$, i.e., x^* is stationary, and*
2. *the Hessian matrix $H(x^*)$ is positive definite, i.e., $s^T H(x^*)s > 0 \quad \forall s \in \mathbb{R}^n \setminus 0$.*

Then x^* is an isolated local minimum of the problem (3.1). More specifically, there exist an $\epsilon > 0$ and a $\mu > 0$ such that

$$f(x) - f(x^*) \geq \frac{\mu}{4}\|x - x^*\|^2.$$

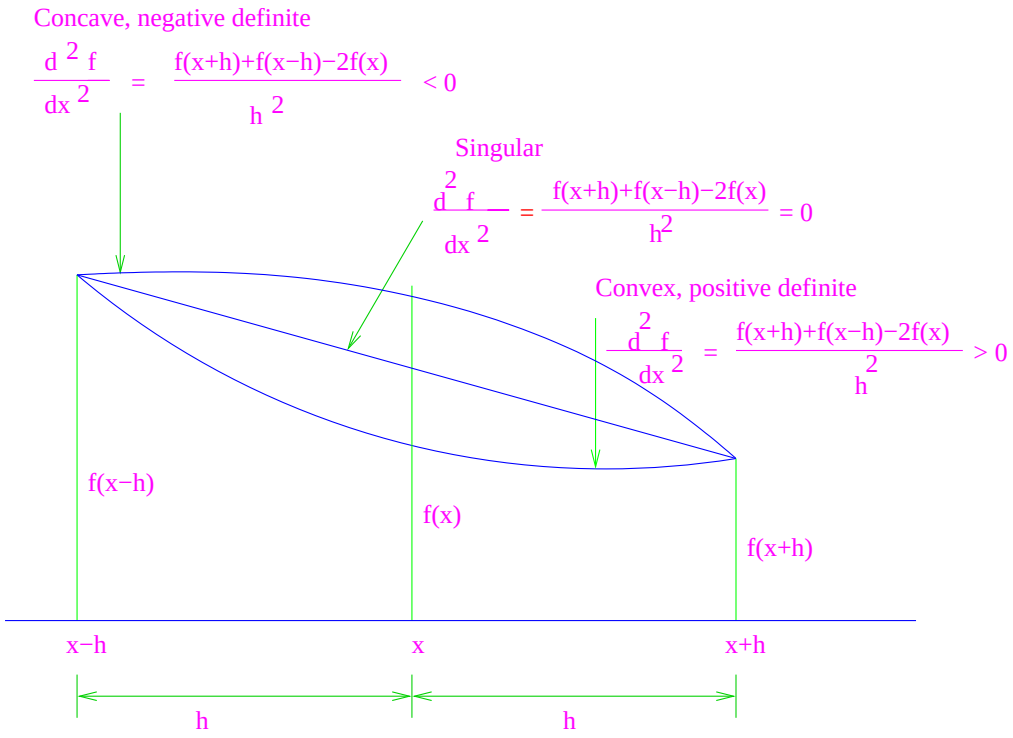

FIGURE 3.2 Convexity and definiteness for a 1-dimensional case.

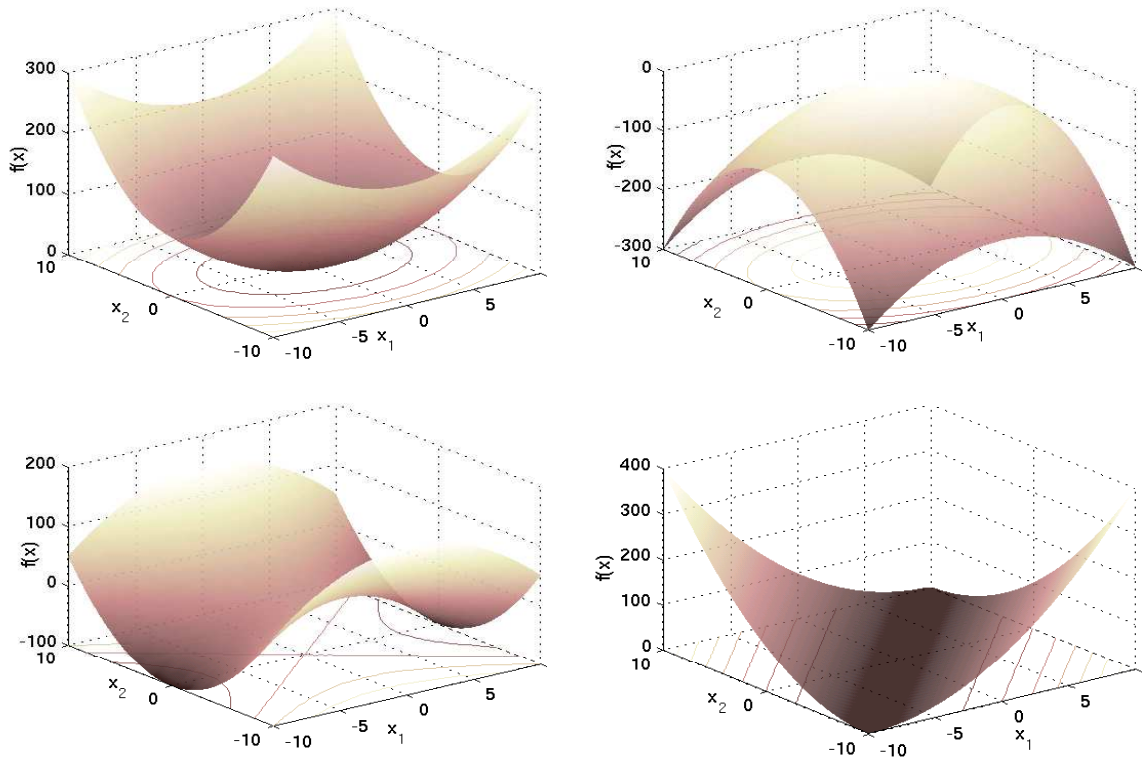

FIGURE 3.3 Figures with different second derivatives.

FIGURE 3.2 illustrates the convexity and definiteness for a 1-dimensional case. FIGURE 3.3 shows some plots with different second derivatives. The Hessian matrix is negative definite at the upper right plot and in this case, the critical point lies at the top (maximum). The Hessian matrix is positive definite at the upper left plot where the critical point lies at the bottom (minimum). The lower left plot in which the Hessian matrix is indefinite has a critical point at the saddle (neither maximum nor minimum). The last plot in which the Hessian matrix is singular has infinitely many critical points (minima).

Therefore, the satisfaction of necessary conditions does not guarantee a minimum. To prove a minimum, the sufficient conditions need to be checked.

3.3 Fundamentals of algorithms

All practical algorithms for solving (3.1) are iterative. Given an initial approximation $x_0 \in \mathbb{R}^n$, a sequence of iterates x_k, $k = 0, 1, 2, \cdots$ is generated in such a way that, hopefully, the approximation to some solution is progressively improved.
The model algorithm for solving optimization problems is presented in Algorithm 3.5. In this algorithm, we start from an initial point x_0. Then a *search direction* s_0 and a *step length* α_0 at x_0 are determined. The search direction should be a descent direction as the aim is to reach a minimum. The step length tells us how far along the search direction our next point x_1 lies so that x_1 is a minimum along s_0. The procedure of computing the step length is called the *line search*. The next point is computed by $x_1 = x_0 + \alpha_0 s_0$. Similarly we compute the next iterates $x_2, x_3, \cdots$ which will successively progress towards a solution.

Algorithm 3.5 ***Model Algorithm [40]***

x_0 = *initial point*
for $k = 0, 1, 2, \cdots$
determine a search direction $s_k \in \mathbb{R}^n$
determine a step length $\alpha_k > 0$ *such that*
$f(x_k + \alpha_k s_k) < f(x_k)$
$x_{k+1} = x_k + \alpha_k s_k$
end

Global convergence

Global convergence [40, p. 20] is guaranteed if Algorithm 3.5 generates a sequence x_k such that

1. $f(x_{k+1}) < f(x_k)$.
2. Every limiting point x^* of x_k is stationary: $g(x^*) = 0$.

First, we will concentrate on the line search methods to compute the step length. In later sections, we will discuss the methods to determine the descent direction.

3.4 Line search methods

The goal of a line search method is to compute a suitable step length α for a given descent direction s_k at x_k by solving

$$\min_{\alpha>0} f(x_k + \alpha s_k). \tag{3.2}$$

Computing the step length α using (3.2) is expensive. There are some methods (e.g., Armijo/Powell-Wolfe line search) to perform the line search in a cheaper way. Before beginning the detail discussion on those methods, we state the following fundamental lemma on line search methods.

Remark: If the level set $N_f(x_0)$ is compact, then

$$M := \max\{\|H(x)\| \quad | \quad x \in N_f(x_0)\} \tag{3.3}$$

is well-defined.

Lemma 3.6 *[44, p. 12] Assume that $x \in N_f(x_0)$ where $N_f(x_0)$ is a compact level set and $\gamma \in (0,1)$, and that $s \in \mathbb{R}^n$ is the descent direction at x. Then there exists a $\tau(x,s,\gamma)$ such that*

1. $f(x+\tau s) - f(x) = \gamma\tau g(x)^T s$
2. $f(x+\alpha s) - f(x) < \gamma\alpha g(x)^T s \qquad \forall\alpha \in (0,\tau)$
3. $\tau \geq 2\frac{(1-\gamma)|g(x)^T s|}{M\|s\|^2} =: \rho$
4. $g(x+\alpha s)^T s < \gamma g(x)^T s \qquad \forall\alpha \in [0, \frac{\rho}{2})$.

3.4.1 Armijo line search

The following condition is known as *Armijo rule*

$$\underbrace{f(x+\alpha s) - f(x)}_{ared} \leq \gamma \underbrace{\alpha g(x)^T s}_{pred} \tag{3.4}$$

with $\gamma \in (0,1)$ which can be obtained from Lemma 3.6. A step length α is accepted when it satisfies the Armijo rule (see FIGURE 3.4). The satisfaction of the Armijo rule can be checked by trying a number of step lengths. This leads to the following algorithm.

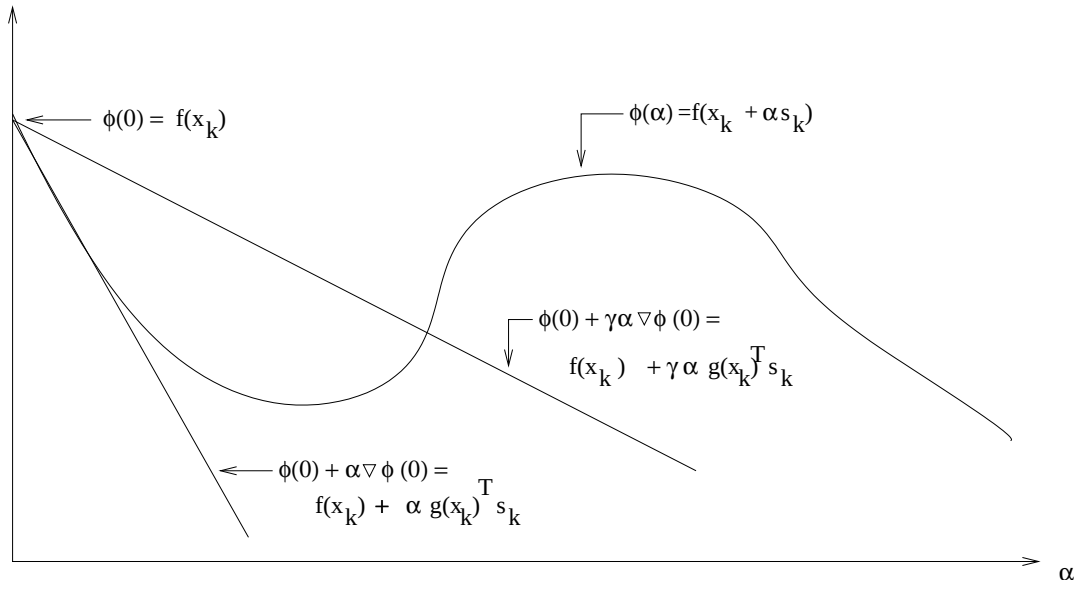

FIGURE 3.4 Armijo line search.

Algorithm 3.7 ***Armijo line search [40]***
Choose $\beta \in (0,1)$ *and* $\gamma \in (0,1)$ *(often* $\gamma \in [10^{-3}, 10^{-2}]$*,* $\beta = 0.5$*). Determine largest step length* $\alpha \in \{1, \beta, \beta^2, \cdots\}$ *from*

$$f(x + \alpha s) - f(x) \leq \gamma \alpha g(x)^T s.$$

There are some important properties of the Armijo rule which we will describe in this subsection. We begin with the following definitions.

Definition 3.8 ***Feasible search direction [40]***
The search direction s_k *of Algorithm 3.5 is called feasible, if*

$$g(x_k)^T s_k < 0, \quad \textit{and} \tag{3.5}$$

$$\lim_{k\to\infty} \frac{g(x_k)^T s_k}{\|s_k\|} = 0 \Rightarrow \lim_{k\to\infty} g(x_k) = 0. \tag{3.6}$$

From (3.5) we deduce that

$$\frac{-g(x_k)^T s_k}{\|s_k\|} = \frac{-g(x_k)^T s_k}{\|g(x_k)\|\|s_k\|}\|g(x_k)\| \tag{3.7}$$

$$= \cos(\angle(-g(x_k), s_k))\|g(x_k\| \tag{3.8}$$

$$\geq c_0\|g(x_k)\|, \tag{3.9}$$

where $c_0 > 0$ is a constant with the property $\cos(\angle(-g(x_k), s_k)) \geq c_0$.

By introducing a strong monotonous growing function $\phi : [0, \infty[\rightarrow [0, \infty[$ with $\phi(0) = 0$ for (3.9) we find that

$$\frac{-g(x_k)^T s_k}{\|s_k\|} \geq \phi(\|g(x_k)\|) \qquad \forall k \geq 0. \tag{3.10}$$

The feasibility of the search direction s_k is guaranteed if conditions (3.6) and (3.9)/(3.10) are satisfied.

Definition 3.9 ***Feasible step length [40]***
The step length α_k of Algorithm 3.5 is called feasible if

$$f(x_k + \alpha s_k) < f(x_k) \qquad \forall k \geq 0 \text{ and} \tag{3.11}$$

$$\lim_{k\rightarrow\infty} f(x_k) - f(x_k + \alpha_k s_k) = 0 \Rightarrow \lim_{k\rightarrow\infty} \frac{g(x_k)^T s_k}{\|s_k\|} = 0. \tag{3.12}$$

Definition 3.10 ***Efficient step length rule [40]***

1. *An efficient step length rule S of is an algorithm which generates a step length $\alpha = S(x, s) > 0$ for every point $x \in N_f(x_0)$ with $g(x) \neq 0$ and every descent search direction $s \in \mathbb{R}^n$.*

2. *A step length rule S is called efficient if there exists a constant $c > 0$ such that $x \in N_f(x_0)$ with $g(x) \neq 0$ and for every generated step length $\alpha = S(x, s) > 0$, the descent direction s satisfies*

$$f(x_k) - f(x_k + \alpha_k s_k) \geq c \left(\frac{g(x_k)^T s_k}{\|s_k\|} \right)^2. \tag{3.13}$$

The Armijo rule is efficient if for a constant $\nu > 0$, the step length satisfies

$$\alpha_k \geq \nu \frac{-g(x_k)^T s_k}{\|s_k\|^2}. \tag{3.14}$$

From the Armijo rule we find that

$$f(x_k) - f(x_k + \alpha_k s_k) \geq -\gamma \alpha_k g(x_k)^T s_k \geq \gamma\nu \left(\frac{g(x_k)^T s_k}{\|s_k\|} \right)^2. \tag{3.15}$$

Setting $c = \gamma\nu$ in (3.15) gives the form (3.13). Therefore, the Armijo rule satisfying (3.14) is an efficient step length rule.

Theorem 3.11 ***Global convergence [40]***
Assume that the Armijo rule is used in Algorithm 3.5 and that for any initial point x_0, the level set $N_f(x_0)$ is compact. If the algorithm generates a descent search direction s_k such that

$$\|s_k\| = \phi\left(\frac{-g(x_k)^T s_k}{\|s_k\|}\right) \qquad \forall k \geq 0, \tag{3.16}$$

where ϕ is a strong monotonous growing function, $\phi : [0,\infty) \to [0,\infty)$, then the Armijo rule generates a feasible step length α_k.

3.4.2 Powell-Wolfe line search

The *Powell-Wolfe rule* to select a step length is given by

$$f(x_k + \alpha_k s_k) - f(x_k) \leq \gamma \alpha_k g(x_k)^T s_k \tag{3.17a}$$

$$g(x_k + \alpha_k s_k)^T s_k \geq \theta g(x_k)^T s_k \tag{3.17b}$$

with constants $0 < \gamma < \theta < 1$. Often $\gamma \in [10^{-3}, 10^{-2}]$, $\theta = 0.9$ are chosen. Note that (3.17) can be derived from Lemma 3.6. We summarize the complete method in Algorithm 3.12.

Algorithm 3.12 ***Powell-Wolfe line search [40]***

1. *Choose $\beta \in (0,1)$ (often $\beta = 0.5$).*
2. *Determine the largest step length $\alpha^- \in \{1, \beta, \beta^2, \cdots\}$ so that $\alpha_k = \alpha^-$ satisfies the Armijo rule. Set $\alpha^+ := \alpha^-/\beta$.*
3. *If $\alpha_k = 1$, determine the smallest step length $\alpha^+ \in \{\beta^{-1}, \beta^{-2}, \cdots\}$ such that $\alpha_k = \alpha^+$ violates the Armijo rule. Set $\alpha^- := \beta\alpha^+$.*
4. *While the condition (3.17b) is violated by $\alpha_k := \alpha^-$,*
   (a) *Compute $\alpha := (\alpha^+ + \alpha^-)/2$.*
   (b) *If (3.17a) is satisfied with $\alpha_k = \alpha$, set $\alpha^- := \alpha$. Otherwise set $\alpha^+ := \alpha$.*
5. *Stop with result $\alpha_k := \alpha^-$.*

The following lemmas give the interesting results from the Powell-Wolfe rule.

Lemma 3.13 *[40] Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and bounded below, i.e.,*

$$\inf_{\alpha>0} f(x + \alpha s) > -\infty$$

for every $x \in \mathbb{R}^n$ with $g(x) \neq 0$ and for every descent direction s. Then there exists a non-empty set $S(x, s) \subset (0, \infty)$ which satisfies the Powell-Wolfe rule with $\alpha \in S(x, s)$.

Lemma 3.14 *[40] Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and bounded below, and g is continuous on the level set $N_f(x_0)$ with an initial point $x_0 \in \mathbb{R}^n$. If the Powell-Wolfe rule is implemented in Algorithm 3.5 and the algorithm generates a sequence of descent search directions s_k, then the Powell-Wolfe rule generates a sequence of feasible and well-defined step lengths α_k.*

We will discuss details of different methodologies to solve the unconstrained optimization problem.

3.5 Steepest descent method

The steepest descent method [30, p. 87] (see Algorithm 3.15) uses the negative gradient of the function $f(x_k)$ as a search direction s_k, since the gradient gives the direction of largest increase of $f(x_k)$. A line search method is employed to obtain a suitable value of the step length α_k along s_k. The solution is updated with these values. At each iteration the search direction is modified as the gradient vector is recalculated.

Algorithm 3.15 ***Steepest descent method*** ***[17]***

x_0 = *initial point*
for $k = 0, 1, 2, \cdots$
$s_k = -g(x_k)$
choose α_k *to minimize* $f(x_k + \alpha_k s_k)$ { *perform line search* }
$x_{k+1} = x_k + \alpha_k s_k$ { *update solution* }
end

In the above algorithm, the computed α minimizes $f(x_k + \alpha s_k)$. So we deduce that

$$\begin{aligned} \frac{df(x_k + \alpha s_k)}{d\alpha} &= s_k^T g(x_k + \alpha s_k) \\ &= s_k^T g(x_{k+1}) = 0 \\ &= s_k^T s_{k+1} = 0, \end{aligned}$$

which implies that successive search directions are orthogonal. This makes the algorithm of steepest descent method slow for certain types of functions as the method searches in the same direction repeatedly.

3.6 Conjugate gradient method

The conjugate gradient method [17, 30, 31] also uses the gradients, but unlike steepest descent method, it does not search in the same directions repeatedly. It modifies the direction at each step to remove components of the previous directions.

The term *conjugate* is defined as follows.

Definition 3.16 ***Conjugate property [31]***
A set of nonzero vectors $s_0, s_1, \cdots, s_n$ is said to be conjugate with respect to a symmetric positive definite matrix B if

$$s_j^T B s_k = 0 \qquad \textit{when } j \neq k. \tag{3.18}$$

For the theoretical framework of the conjugate gradient method, we assume the function to be minimized has a quadratic form

$$f(x) = c + b^T x + \frac{1}{2} x^T B x. \tag{3.19}$$

Here B is an $n \times n$ symmetric positive definite matrix, b is a vector of length n, and c is a constant. We start from a point x_0 and minimize $f(x)$ successively along n linearly independent directions $s_0, s_1, s_2, \cdots, s_n$. We construct the directions $s_0, s_1, s_2 \cdots$, which are mutually conjugate with respect to B of (3.19). At each stage k, the direction s_{k+1} is obtained from the linear combination of the gradient g_{k+1} and the previous search direction s_k:

$$s_{k+1} = -g_{k+1} + \beta_k s_k, \tag{3.20}$$

with $g_k := g(x_k)$. The coefficient β_k is chosen in a way that s_{k+1} becomes conjugate to

all the previous directions. By multiplying (3.20) by $s_k^T B$, we have

$$\begin{aligned} & s_k^T B s_{k+1} = s_k^T B(-g_{k+1} + \beta_k s_k) \\ \Rightarrow \quad & 0 = -s_k^T B g_{k+1} + \beta_k s_k^T B s_k \\ \Rightarrow \quad & \beta_k = \frac{g_{k+1}^T B s_k}{s_k^T B s_k}. \end{aligned}$$

These useful properties of the conjugate gradient method lead to the following algorithm.

Algorithm 3.17 ***Conjugate gradient method with modification from Polak and Ribiere [17]***

x_0 = *initial point*
$g_0 = g(x_0)$
$s_0 = -g_0$
for $k = 0, 1, 2, \cdots$
  choose α_k *to minimize* $f(x_k + \alpha_k s_k)$ { *perform line search* }
  $x_{k+1} = x_k + \alpha_k s_k$ { *update solution*}
  $g_{k+1} = g(x_{k+1})$ { *new gradient*}
  $\beta_k = ((g_{k+1} - g_k)^T g_{k+1})/(g_k^T g_k)$
  $s_{k+1} = -g_{k+1} + \beta_k s_k$ { *new search direction*}
end

We give the following theorems which clarify details of Algorithm 3.17.

Theorem 3.18 *[31] Let x_0 be an initial point and assume that the sequence $\{x_k\}$ is generated by Algorithm 3.17. Then*

$$g_k^T s_l = 0, \quad l = 0, \cdots, k-1,$$

and x_k is the minimum of (3.19) over the set $\{x \mid x = x_0 + span\{s_0, s_1, \cdots, s_{k-1}\}\}$.

Theorem 3.19 *[30, p. 92] At any stage k of Algorithm 3.17 where $g_i \neq 0, i = 0, 1, \cdots, k$ we have*

$$\alpha_k = \frac{g_k^T g_k}{s_k^T B s_k} \neq 0, \tag{3.21}$$

$$\beta_k = \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k}. \tag{3.22}$$

$$= \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}. \tag{3.23}$$

Theoretically, with the conjugate gradient method, the minimum of a quadratic objective function of dimension n is achieved in most n iterations. The exact line search requires the matrix B which is actually the Hessian matrix H and is expensive to compute. In case of a negative H the exact line search will fail. In practice, for general nonlinear functions, Algorithm 3.17 is implemented with the Powell-Wolfe rule instead of exact line search mentioned in Theorem 3.19. But this algorithm with Powell-Wolfe rule does not always guarantee that s_k is a descent direction. However, the descent property holds when we choose

$$\beta_k = \max\{\beta_k, 0\}.$$

Another modification of nonlinear conjugate gradient method is to restart the iteration after every n steps by setting $\beta_k = 0$. This restarting refreshes the algorithm, deleting old information that may not be required [31, p. 103].

3.7 Newton's method

The objective function is locally approximated by a quadratic function which can be obtained from the truncated Taylor series expansion

$$f(x + s) \approx f(x) + g^T s + \frac{1}{2} s^T H(x) s, \tag{3.24}$$

where $s \in \mathbb{R}^n$ is the search direction. s will be called the *Newton step* in the local Newton method. The quadratic function (3.24) is minimized in s when

$$H(x)s = -g. \tag{3.25}$$

The algorithm for the local Newton method can be given as follows. We will use the notation $g_k = g(x_k)$ and $H_k = H(x_k)$.

Algorithm 3.20 ***Local Newton method [17]***

x_0 = *initial point*

for $k = 0, 1, 2, \cdots$

solve $H_k s_k = -g_k$ *for* s_k

$x_{k+1} = x_k + s_k$ {*update solution*}

end

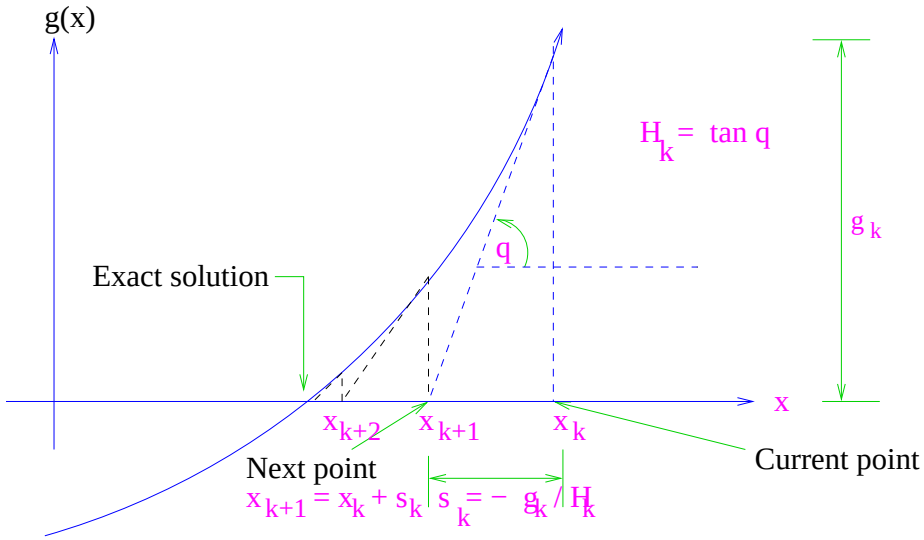

FIGURE 3.5 Newton's method for solving a 1-dimensional nonlinear optimization problem.

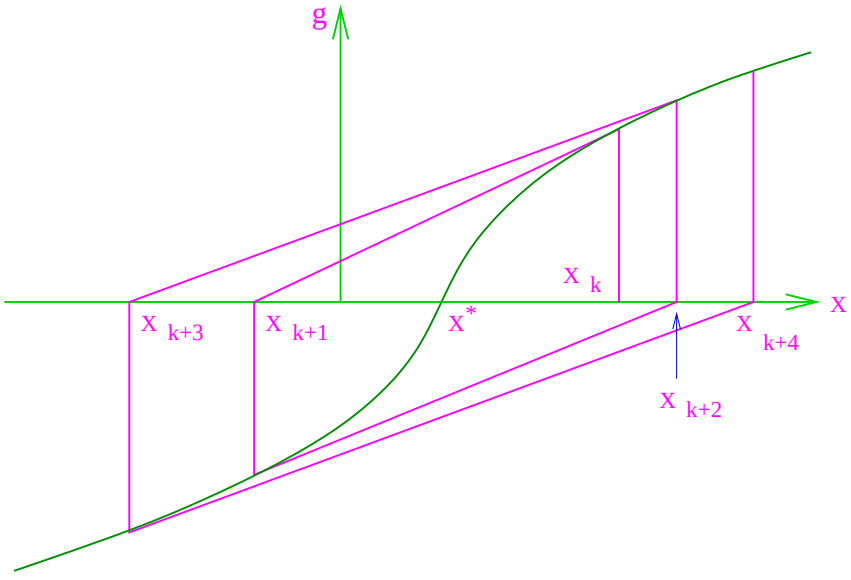

FIGURE 3.6 An example where the local Newton method fails.

FIGURE 3.5 shows how Newton's method (Algorithm 3.20) converges to a solution. The method usually converges provided that the initial point is close to a solution. Theorem 3.21 gives the local convergence result.

Theorem 3.21 *[40] If $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable and $x^* \in \mathbb{R}^n$ satisfies the second order sufficient condition of a local minimum, then the following criteria hold true for a constant $\epsilon > 0$ and for all $x \in \mathbb{B}_\epsilon(x^*)$.*

1. *$\nabla^2 f(x)$ is positive definite with*

$$\|H(x)^{-1}\| \leq 2\|H(x^*)^{-1}\| = \frac{2}{\lambda_{min}(H(x^*))}.$$

2. *x^* is an isolated local minimum of f in $\mathbb{B}_\epsilon(x^*)$ and the following condition holds.*

$$\frac{\lambda_{min}(H(x^*))}{2}\|x - x^*\| \leq \|g(x)\| \leq 2\lambda_{max}(H(x^*))\|x - x^*\|.$$

3. *If H is Lipschitz continuous in $\mathbb{B}_\epsilon(x^*)$ with a constant L, then the sequence $\{x_k\} \subset \mathbb{B}_\epsilon(x^*)$ converges Q-quadratically to x^* where*

$$\|x_{k+1} - x^*\| \leq \frac{L}{\lambda_{min}(H(x^*))}\|x_k - x^*\|^2 \quad \text{for all } k \geq 0.$$

The method may fail when started far from the solution. In FIGURE 3.6 the iterates x_k of the local Newton method are moving away from the solution x^* which indicates that the method diverges. The possible cause of failure of the local Newton method is that Hessian matrices are close to singular or indefinite which may generate excessively long or ascent search directions s_k.

Newton's method requires substantial amount of work per iteration, for a problem with dense Hessian matrix, each iteration requires $O(n^2)$ scalar function evaluations to form gradient vector and Hessian matrix and $O(n^3)$ arithmetic operations to solve the linear system for the Newton step [17, chap. 5].

To obtain global convergence, the search direction s_k must be descent. When the Hessian matrix H_k is positive definite, s_k is a descent direction. When H_k is singular, we use an approximate matrix $B_k \in \mathbb{R}^{n\times n}$ of H_k where B_k is formulated from H_k and is positive definite for each k. Moreover, s_k could be controlled using a line search method when s_k becomes excessively long. This leads to the following algorithm.

Algorithm 3.22 ***Global Newton method*** *[40, p. 45]*
Choose constants $\gamma \in (0, 1/2), c_1 \in (0,1), c_2 \in (0,1)$, $\nu > \mu > 0$ *and* $p > 0$

x_0 = *initial point*
for $k = 0, 1, 2, \cdots$
  Solve $H_k s_k = -g_k$ *for* s_k
  if $-g_k^T s_k < \min(c_1, c_2 \|g_k\|^p)\|g_k\|\|s_k\|$,
    Choose a positive definite matrix $B_k \in \mathbb{R}^{n\times n}$ *with*
    $\lambda_{min}(B_k) \geq \mu,\ \lambda_{max}(B_k) \leq \nu$;
    Solve $B_k s_k = -g_k$ *for* s_k.
  Determine α_k *by the Armijo rule*
  $x_{k+1} = x_k + \alpha_k s_k$ {*update solution*}
end

Remark: The matrix B_k is very often selected from the Hessian matrix H_k with a $\mu > 0$ as

$$B_k = H_k + (\mu + max(0, -\lambda_{min}(H_k)))I$$

to avoid singularity or indefiniteness. We have the following result for the global Newton method.

Theorem 3.23 *[40] If* $f : \mathbb{R}^n \to \mathbb{R}$ *is twice continuously differentiable and* $N_f(x_0)$ *is compact, Algorithm 3.22 generates a sequence of feasible search directions* s_k *and a sequence of feasible step lengths* α_k.

3.8 Quasi-Newton method

The quasi-Newton method is similar to Newton's method, but it uses an approximate Hessian matrix B_k instead of the true Hessian matrix H_k, and thus the cost per iteration reduces. In this method, only gradient evaluation is required at each iteration. The Hessian matrix is often initially approximated by a unit matrix I. The next approximate Hessian matrix B_{k+1} is calculated using the gradients $g_k,\ g_{k+1}$, the step s_k and the current approximate Hessian matrix B_k [17, chap. 5] which we will discuss now.

From the Taylor series expansion we find that

$$g_{k+1} = g_k + H_k(x_{k+1} - x_k) + o(x_{k+1} - x_k).$$

Ignoring higher order terms and writing H_k as B_{k+1}, we obtain

$$\begin{aligned} &\Rightarrow \quad B_{k+1}(x_{k+1} - x_k) = g_{k+1} - g_k \\ &\Rightarrow \qquad B_{k+1}s_k = \underbrace{g_{k+1} - g_k}_{y_k}. \end{aligned} \tag{3.26}$$

Hence, B_{k+1} can be approximated by a function ξ of $B_k,\ s_k$ and y_k which we will elaborate in the next subsection.

$$B_{k+1} = \xi(B_k, s_k, y_k). \tag{3.27}$$

The following algorithm gives the fundamental form of the quasi-Newton method.

Algorithm 3.24 ***Local quasi-Newton method [40]***

x_0 = *initial point*
B_0 = *initial approximate Hessian*
for $k = 0, 1, 2, \cdots$
  solve $B_k s_k = -g_k$ *for* s_k
  $x_{k+1} = x_k + s_k$ {*update solution*}
  $B_{k+1} = \xi(B_k, s_k, y_k)$ {*update approximate Hessian matrix*}
end

3.8.1 Update of approximate Hessian matrix

There are several formulas [40, p. 63–67] to update the approximate Hessian matrix. We will discuss the different updates which can be given by

$$B_{k+1} = B_k + \alpha_k v_k v_k^T + \beta_k w_k w_k^T, \qquad \alpha_k, \beta_k \in \{\pm 1\}$$

where $\alpha_k,\ \beta_k,\ v_k,\ w_k$ are chosen such that B_{k+1} satisfies (3.26).

Davidson-Fletcher-Powell (DFP) update

The matrix B_{k+1} takes the following form with the DFP update

$$\begin{aligned} B_{k+1}^{DFP} = \xi^{DFP}(B_k, s_k, y_k) \quad &= B_k + \frac{(y_k - B_k s_k)y_k^T + y_k(y_k - B_k s_k)^T}{y_k^T s_k} - \frac{(y_k - B_k s_k)^T s_k}{(y_k^T s_k)^2} y_k y_k^T \\ &= \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right)^T B_k \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) + \frac{y_k y_k^T}{y_k^T s_k}. \end{aligned}$$

The matrix B_{k+1}^{DFP} is symmetric and very often well-defined if $y_k^T s_k \neq 0$. We can prove that the matrix satisfies the quasi-Newton equation (3.26).

Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

The matrix B_{k+1}^{BFGS} takes the following form with the BFGS update

$$B_{k+1}^{BFGS} = \xi^{BFGS}(B_k, s_k, y_k) = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}.$$

The matrix B_{k+1}^{BFGS} is symmetric and very often well-defined if $y_k^T s_k \neq 0$ and $s_k^T B_k s_k \neq 0$. It can be verified that the matrix satisfies the quasi-Newton equation (3.26).

Powell's Symmetric Broyden (PSB) update

The matrix B_{k+1}^{PSB} takes the following form with the PSB update

$$B_{k+1}^{PSB} = \xi^{PSB}(B_k, s_k, y_k) = B_k + \frac{(y_k - B_k s_k)s_k^T + s_k(y_k - B_k s_k)^T}{s_k^T s_k} - \frac{(y_k - B_k s_k)^T s_k}{(s_k^T s_k)^2} s_k s_k^T.$$

The matrix B_{k+1}^{PSB} is symmetric and very often well-defined if $s_k \neq 0$. This matrix satisfies the quasi-Newton equation (3.26).

Lemma 3.25 ***B_k^{DFP} and B_k^{BFGS} are symmetric and invertible [40]***

1. *B_{k+1}^{DFP} and B_{k+1}^{BFGS} are invertible and the following conditions hold true if $y_k^T s_k \neq 0, s_k^T B_k s_k \neq 0$ and $y_k^T B_k^{-1} y_k \neq 0$.*

$$\left(B_{k+1}^{DFP}\right)^{-1} = \xi^{BFGS}(B_k^{-1}, y_k, s_k), \left(B_{k+1}^{BFGS}\right)^{-1} = \xi^{DFP}(B_k^{-1}, y_k, s_k),$$

2. *If B_k is symmetric positive definite and $y_k^T s_k > 0$, then $B_{k+1}^{DFP}, B_{k+1}^{BFGS}$ are positive definite.*

3.8.2 Update of inverse of approximate Hessian matrix

In practice, in the quasi-Newton method, the inverse of the approximate Hessian matrix B_k is updated, rather than B_k itself, so that the quasi-Newton step s_k can be solved with a cost per iteration of $O(n^2)$ rather than $O(n^3)$ operation. We have the following update formulas [40, p. 67–68] for the inverse of an approximate Hessian matrix.
BFGS version:

$$\begin{aligned} M_{k+1} &= \left(B_{k+1}^{BFGS}\right)^{-1} = \xi^{DFP}(M_k, y_k, s_k) \\ &= M_k + \frac{(s_k - M_k y_k)s_k^T + s_k(s_k - M_k y_k)^T}{y_k^T s_k} - \frac{(s_k - M_k y_k)^T y_k}{(y_k^T s_k)^2} s_k s_k^T. \end{aligned} \quad (3.28)$$

DFP version:

$$\begin{aligned} M_{k+1} &= \left(B_{k+1}^{DFP}\right)^{-1} = \xi^{BFGS}(M_k, y_k, s_k) \\ &= M_k + \frac{s_k s_k^T}{y_k^T s_k} - \frac{B_k y_k y_k^T B_k}{y_k^T M_k y_k}. \end{aligned} \tag{3.29}$$

3.8.3 Global convergence of BFGS method

In the following BFGS algorithm we incorporate the quasi-Newton method with the Powell-Wolfe rule.

Algorithm 3.26 ***Global BFGS method [40]***

x_0 = *initial point*
M_0 = *initial positive definite matrix such that* $M_0 = B_0^{-1}$
for $k = 0, 1, 2, \cdots$
Compute $s_k = -M_k g_k$
Determine α_k *with the Powell-Wolfe line search*
$x_{k+1} = x_k + \alpha_k s_k$ {*update solution*}
Compute $M_{k+1} = \xi^{DFP}(M_k, y_k, \alpha_k s_k)$ *by (3.28)*
end

Now we describe the useful properties of the BFGS method in the following theorems.

Theorem 3.27 *[40] Algorithm 3.26 generates a sequence of matrices* $B_k = M_k^{-1}$ *which are symmetric positive definite and satisfy* $y_k^T s_k > 0$ *and every generated* s_k *is a descent direction.*

Theorem 3.28 ***Global convergence of BFGS method [40]***
Assume that $f : \mathbb{R}^n \to \mathbb{R}$ *is twice continuously differentiable and the level set* $N_f(x_0)$ *is convex and* f *is convex on* $N_f(x_0)$ *with a bounded Hessian matrix* H *such that*

$$\mu\|s\|_2 \le s^T H s \le \nu\|s\|_2 \qquad \forall x \in N_f(x_0), \forall s \in \mathbb{R}^n$$

with constants $0 < \mu \le \nu$. *Then Algorithm 3.26 generates a sequence of points* x_k *which converges to a unique minimum* x^* *of* f.

Now we present the following theorem which states that the BFGS method gives superlinear convergence rate.

Theorem 3.29 *[40] Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuous differentiable. BFGS update is used for the matrices B_k and the generated sequence of iterates converges to x^* which satisfies the second order sufficient condition and H is Lipschitz continuous at x^*, then the sequence converges at a superlinear rate and the condition holds true*

$$\lim_{k\to\infty} \|B_{k+1} - H_k\| \to 0.$$

3.9 Trust-region method

Trust-region methods [7, 10, 31] define a region around the *current iterate* x_k which they trust to be representable by a model of $f(x_k + s)$. For this approach, the model $m_k(s)$ is constructed using linearization or any Taylor series approximation of the function $f(x_k + s)$. This model $m_k(s)$ represents $f(x_k + s)$ well in a certain region around x_k, the so-called *trust-region*. The step s_k is computed to reduce the model of $f(x_k + s)$. If the reduction predicted by $m_k(s_k)$ is sufficiently realized by $f(x_k + s_k)$, the next iterate is calculated by the formula $x_{k+1} = x_k + s_k$. Otherwise, $x_{k+1} = x_k$ is set and the model is refined by reducing the *trust-region radius* Δ. In practical algorithms, the trust-region radius is adjusted between iterations according to the agreement between predicted and actual reduction in the objective function.

We consider a quadratic model of $f(x_k + s)$ by the Taylor series approximation:

$$m_k(s) = f_k + {g_k}^T s + \frac{1}{2} s^T B_k s, \tag{3.30}$$

where B_k is a symmetric matrix which approximates the local Hessian matrix H_k. In most practical cases, these models are unlikely to represent $f(x_k + s)$ if s is large. Moreover, the models may be unbounded from below so that any attempts to minimize them may generate a large step. This problem will always arise if B_k is indefinite or singular. To prevent the model $m_k(s)$ from being unboundedness we impose a trust-region constraint

$$\|s\| \leq \Delta_k$$

with $\Delta_k > 0$, on the step.

Now we note that

$$f(x_k + s) = f_k + {g_k}^T s + \frac{1}{2} s^T H(x_k + ts)s,$$

for some scalar $t \in (0, 1)$. Since $m_k(s) = f_k + {g_k}^T s + O(\|s\|^2)$, the approximation error is given by

$$|f(x_k + s) - m_k(s)| = O(\|s\|^2).$$

The idea of imposing a trust-region constraint comes from the above result which implies that the approximation error $|f(x_k + s) - m_k(s)|$ can be decreased by restricting the allowable step. Finally, our *trust-region subproblem* is given by

$$\min_{s \in \mathbb{R}^n} m_k(s) \text{ subject to } \|s\| \leq \Delta_k,$$

and we shall choose s_k as an approximate solution of this problem.

Algorithm 3.30 ***Trust-region method [16]***
Given $k = 0$, $\Delta_0 > 0$ and x_0, until "convergence" do:
Build the quadratic model $m_k(s)$ of $f(x_k + s)$.
Solve the trust-region subproblem for s_k.
Define

$$\rho_k = \frac{f_k - f(x_k + s_k)}{f_k - m_k(s_k)}.$$

If $\rho_k \geq \gamma_v$ [very successful] $\boxed{0 < \gamma_v < 1}$
set $x_{k+1} = x_k + s_k$ and $\Delta_{k+1} = \mu_i \Delta_k$. $\boxed{\mu_i \geq 1}$
Otherwise if $\rho_k \geq \gamma_s$ then [successful] $\boxed{0 < \gamma_s \leq \gamma_v < 1}$
set $x_{k+1} = x_k + s_k$ and $\Delta_{k+1} = \Delta_k$.
Otherwise [unsuccessful]
set $x_{k+1} = x_k$ and $\Delta_{k+1} = \mu_r \Delta_k$. $\boxed{0 < \mu_r < 1}$
Increase k by 1.

3.9.1 Basic trust-region method

As mentioned before, we accept $x_{k+1} = x_k + s_k$ whenever the predicted model reduction $f_k - m_k(s_k)$ is sufficiently realized by the actual reduction $f_k - f(x_k + s_k)$. To measure this the *reduction ratio*

$$\rho_k = \frac{f_k - f(x_k + s_k)}{f_k - m_k(s_k)}$$

of actual to predicted reduction is computed. The step s_k is accepted when ρ_k is not unacceptably smaller than 1.0. If the ratio is close to (or larger than) 1.0, we increase the radius as in this case, future step computations will hopefully benefit from an increase in the trust-region radius. On the other hand, if ρ is much smaller than 1.0, i.e. the agreement between the actual and predicted reduction is poor (or particularly, f actually increases), the current step is rejected. In this case, we reduce the trust-region radius to find a more suitable step at the next iteration.

We present the basic trust-region method in Algorithm 3.30. Reasonable values of the constants might be $\gamma_v = 0.9$ or 0.99, $\gamma_s = 0.1$ or 0.01, $\mu_i = 2$, and $\mu_r = 0.5$.

We check that it makes a little justification in increasing the trust-region radius following a very successful iteration unless $\|s_k\| \approx \Delta_k$, and in decreasing the radius less than what is required to discard an unsuccessful s_k. In practice, for a very successful iteration,

if the step is much shorter (say less than half the trust-region radius), the trust-region radius is not increased. There exist various schemes for choosing an initial trust-region radius. However, For a well-scaled problem, $\Delta_0 = O(1)$ is reasonable. Poor scaling can affect the performance of trust-region methods.

3.9.2 Basic convergence of trust-region methods

Trust-region methods have very strong convergence properties like steepest-descent methods. Now we check that we at least achieve as much reduction in the model as we would from an iteration of steepest descent. We define the *Cauchy* point as $s_k^C = -\alpha_k^C g_k$, where

$$\begin{aligned} \alpha_k^C &= \arg\min_{\alpha>0} m_k(-\alpha g_k) \text{ subject to } \alpha\|g_k\| \leq \Delta_k \\ &= \arg\min_{0<\alpha\leq\Delta_k/\|g_k\|} m_k(-\alpha g_k). \end{aligned}$$

We shall require that our step s_k satisfies

$$m_k(s_k) \leq m_k(s_k^C) \text{ and } \|s_k\| \leq \Delta_k. \tag{3.31}$$

The Cauchy point can be easily computed, since it merely requires that we minimize the quadratic model along a line segment. Trust-region methods have far better convergence properties than steepest descent methods. We now check the convergence of our trust-region method. We shall use the scalar $\kappa_l \geq \kappa_s > 0$ with $\kappa_l\|g_k\| \geq \|g_k\|_2 \geq \kappa_s\|g_k\|$.

With the following theorem, we guarantee a reasonable reduction in the trust-region model at the Cauchy point.

Theorem 3.31 *[10] If s_k^C is the Cauchy point of the quadratic model $m_k(s)$ within the trust-region $\|s\| \leq \Delta_k$, then*

$$f_k - m_k(s_k^C) \geq \frac{1}{2}\|g_k\|_2 \min\left[\frac{\|g_k\|_2}{1+\|B_k\|_2}, \kappa_s\Delta_k\right].$$

This shows that the guaranteed reduction depends on the norm of the current gradient, and is also affected by the size of both the trust-region radius and the (inverse of the) Hessian.

The following corollary states that the step in our algorithm is at least the Cauchy point.

Corollary 3.32 *[35] If s_k is an improvement on the Cauchy point of the quadratic model $m_k(s)$ within the trust-region $\|s\| \leq \Delta_k$,*

$$f_k - m_k(s_k) \geq \frac{1}{2}\|g_k\|_2 \min\left[\frac{\|g_k\|_2}{1+\|B_k\|_2}, \kappa_s\Delta_k\right].$$

This gives a typical trust-region result which relates the model reduction to a measure of the distance to the minimum, in this case measured in terms of the norm of the gradient.

Observe that we are using the quadratic model $m_k(s)$ for which the first two terms are exactly those from the Taylor's series approximation, and therefore, the difference between model and function will vary like the square of the norm of s_k which is described in the following lemma.

Lemma 3.33 *[16] Suppose that f is twice continuously differentiable, and that the true and model Hessians satisfy the bounds $\|H(x)\|_2 \leq \kappa_h$ for all x and $\|B_k\|_2 \leq \kappa_b$ for all k and some $\kappa_h \geq 1$ and $\kappa_b \geq 0$. Then*

$$|f(x_k + s_k) - m_k(s_k)| \leq \kappa_d \Delta_k^2,$$

where $\kappa_d = \frac{1}{2}\kappa_l^2(\kappa_h + \kappa_b)$, for all k.

We therefore notice that the error between the objective function and the model decreases quadratically with the trust-region radius. This result shows that there must be good local agreement between the model and objective functions, if the trust-region radius is small enough. However, rather than requiring a uniform bound on $H(x)$, all that is actually needed is a similar bound for all x between x_k and $x_k + s_k$.

The next lemma provides a crucial result, in that it will always be possible to make progress from a non-stationary point ($g_k \neq 0$).

Lemma 3.34 *[16] Suppose that f is twice continuously differentiable, and that the true and model Hessians satisfy the bounds $\|H_k\|_2 \leq \kappa_h$ and $\|B_k\|_2 \leq \kappa_b$ for all k and some $\kappa_h \geq 1$ and $\kappa_b \geq 0$, and that $\kappa_d = \frac{1}{2}\kappa_l^2(\kappa_h + \kappa_b)$. Suppose furthermore that $g_k \neq 0$ and that*

$$\Delta_k \leq \|g_k\|_2 \min\left(\frac{1}{\kappa_s(\kappa_h + \kappa_b)}, \frac{\kappa_s(1-\gamma_v)}{2\kappa_d}\right),$$

with the notation from Algorithm 3.30. Then iteration k is very successful and

$$\Delta_{k+1} \geq \Delta_k.$$

As a consequence of this property, we will prove that the radius is uniformly bounded away from zero if the same is true for the sequence of gradients, that is the radius will not shrink to zero at non-stationary points.

Lemma 3.35 *[16] Suppose that f is twice continuously differentiable and that the true and model Hessians satisfy the bounds $\|H_k\|_2 \leq \kappa_h$ and $\|B_k\|_2 \leq \kappa_b$ for all k and some $\kappa_h \geq 1$ and $\kappa_b \geq 0$, and that $\kappa_d = \frac{1}{2}\kappa_l^2(\kappa_h + \kappa_b)$. Suppose furthermore that there exists a constant $\epsilon > 0$ such that $\|g_k\|_2 \geq \epsilon$ for all k. Then*

$$\Delta_k \geq \kappa_\epsilon := \epsilon\mu_r \min\left(\frac{1}{\kappa_s(\kappa_h + \kappa_b)}, \frac{\kappa_s(1-\gamma_v)}{2\kappa_d}\right) \quad \forall k$$

with the notation from Algorithm 3.30.

This property is crucial to ensure the progress of the algorithm (except x_k lying at a critical point).

The preceding two results are sufficient to establish that if there is only a finite number of successful iterations, the iterates must converge to a critical point after the last of these.

Lemma 3.36 *[16] Suppose that f is twice continuously differentiable, and that both the true and model Hessians remain bounded for all k. Suppose furthermore that there are only finitely many successful iterations. Then $x_k = x^*$ for all sufficiently large k and $g(x^*) = 0$.*

Now we consider the case where there are infinitely many successful iterations. In this case, we will prove that the global convergence guarantees that there is at least one sequence of gradients that converges to zero.

Theorem 3.37 *[16] Suppose that f is twice continuously differentiable, and that both the true and model Hessians remain bounded for all k. Then either*

$$g_l = 0 \text{ for some } l \geq 0$$

or

$$\lim_{k\to\infty} f_k = -\infty$$

or

$$\liminf_{k\to\infty} \|g_k\| = 0.$$

We now prove the stronger result that it is not possible for a second sub-sequence of gradients to stay bounded away from zero.

Corollary 3.38 *[16] Suppose that f is twice continuously differentiable, and that both the true and model Hessians remain bounded for all k. Then either*

$$g_l = 0 \text{ for some } l \geq 0$$

or

$$\lim_{k \to \infty} f_k = -\infty$$

or

$$\lim_{k \to \infty} \|g_k\| = 0.$$

Hence, we arrive at a crucial result that the gradients of the sequence $\{x_k\}$ generated by our algorithm converge to, or are all ultimately, zero. This does not mean that a subsequence of $\{x_k\}$ itself converges, but if it does, the limit is a critical point.

We have seen that trust-region methods have very powerful convergence theories. We now turn to the practical issues, namely how one can find a suitable step s_k.

3.9.3 Solving the trust-region subproblem

We seek the solution s^* of the minimization problem

$$\min_{s \in \mathbb{R}^n} m(s) \equiv g^T s + \frac{1}{2} s^T B s \text{ subject to } \|s\| \leq \Delta. \tag{3.32}$$

For brevity, we have dropped the iteration subscript k and discarded the constant term f from (3.30) to formulate the above problem. We shall only consider the 2-norm trust-region $\|s\|_2 \leq \Delta$, mainly because there are very efficient and powerful methods in this case, but of course other norms are possible and are sometimes preferred in practice. A simple speculation suggests how we might solve the above trust-region problem. First, we find the unconstrained minimum of the model. If the minimum lies outside the trust-region, the model minimum must occur on the trust-region boundary and thus can be found as the global minimum of $m(s)$ subject to $\|s\|_2 \leq \Delta$.

We have a strong solution characterisation result for the 2-norm trust-region subproblem.

Theorem 3.39 *[10] Any* global *minimum s^* of $m(s)$ subject to $\|s\|_2 \leq \Delta$ satisfies the equation*

$$(B + y^* I)s^* = -g,$$

where $B + y^* I$ is positive semi-definite, $y^* \geq 0$ and $y^*(\|s^*\|_2 - \Delta) = 0$. If $B + y^* I$ is positive definite, s^* is unique.

This theorem gives necessary and sufficient *global* optimality conditions for a nonconvex optimization problem (that is, a problem which may have a number of local minima). We

notice that the necessary and sufficient conditions are identical. Trust-region subproblem can be solved using these optimality conditions.

We have two cases to consider. If B is positive definite and the solution s to

$$Bs = -g \tag{3.33}$$

satisfies $\|s\|_2 \leq \Delta$, then it immediately follows that $s^* = s$ ($y^* = 0$ in Theorem 3.39). This potential solution can simply be checked by the Cholesky decomposition (section 2.8.1) of B. If the Cholesky decomposition is successful, using this decomposition, (3.33) is solved and subsequently $\|s\|_2$ is evaluated. Otherwise, either B is positive definite but the solution to (3.33) satisfies $\|s\|_2 > \Delta$ or B is singular or indefinite. In these cases, Theorem 3.39 implies that s satisfies

$$(B + yI)s = -g \text{ and } s^T s = \Delta^2, \tag{3.34}$$

which is a *nonlinear* (quadratic) system of algebraic equations with $n + 1$ unknowns s and y. Now we try to solve this system. Suppose B has an eigendecomposition

$$B = U^T \Lambda U,$$

where Λ is a diagonal matrix of eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and U is an orthonormal matrix of associated eigenvectors. The first part of (3.34) enables us to write s explicitly in terms of y, that is

$$s(y) = -(B + yI)^{-1} g.$$

Now we substitute $s(y)$ into the second part of (3.34) which gives

$$\|s(y)\|_2^2 = \|U^T(\Lambda + yI)^{-1}Ug\|_2^2 = \sum_{i=1}^{n} \frac{\gamma_i^2}{(\lambda_i + y)^2} = \Delta^2, \tag{3.35}$$

where $\gamma_i = {e_i}^T(Ug) = (e_i^T U)g$.

When B is positive definite and $\|s\|_2 > \Delta$, there is a strictly positive $y \in (0, \infty)$ such that $\|s(y)\|_2^2 = \Delta^2$.

Now we concentrate on the case where B is indefinite or singular. Theorem 3.39 requires that $B + yI$ be positive semi-definite. Therefore, the solution (s, y) to (3.34) necessarily satisfies $y \geq -\lambda_1$. When there exists a $y > -\lambda_1$ such that $\|s(y)\|_2 = \Delta$, the situation is referred to as the *easy* case. Then $B + yI$ is positive definite and thus (3.34) has a unique solution.

Thus to solve the trust-region subproblem, we have to find a particular root of the nonlinear equation $\|s(y)\|_2 = \Delta$. We could use Newton's method to solve $\|s(y)\|_2 - \Delta = 0$ for y. But $s(y)$ is highly nonlinear close to $-\lambda_1$. So Newton's method will be unreliable or very slow in this case. Therefore we solve an equivalent equation

$$\phi(y) := \frac{1}{\|s(y)\|_2} - \frac{1}{\Delta} = 0, \tag{3.36}$$

as it is nearly linear near the optimal y. The Newton step of this equation is given by $-\phi(y)/\phi'(y)$. Now we shall expand our observation on $\phi'(y)$. Since

$$\phi(y) = \frac{1}{(s(y)^T s(y))^{\frac{1}{2}}} - \frac{1}{\Delta},$$

it follows, on differentiating, that

$$\phi'(y) = -\frac{s(y)^T \nabla_y s(y)}{(s(y)^T s(y))^{\frac{3}{2}}} = -\frac{s(y)^T \nabla_y s(y)}{\|s(y)\|_2^3}.$$

In addition, on differentiating the defining equation

$$(B + yI)s(y) = -g,$$

it must be that

$$(B + yI)\nabla_y s(y) + s(y) = 0.$$

Notice that, rather than the value of $\nabla_y s(y)$, merely the numerator

$$s(y)^T \nabla_y s(y) = -s(y)^T (B + yI)^{-1} s(y)$$

is required in the expression for $\phi'(y)$. Given the Cholesky decomposition (section 2.8.1) $B + yI = L(y)L(y)^T$, we deduce that

$$s(y)^T (B + yI)^{-1} s(y) = s(y)^T L(y)^{-T} L(y)^{-1} s(y) = \left[L(y)^{-1} s(y)\right]^T \left[L(y)^{-1} s(y)\right] = \|w(y)\|_2^2,$$

where $L(y)w(y) = s(y)$.
Using above results the Newton step is simplified to the form

$$\begin{aligned} -\frac{\phi(y)}{\phi'(y)} &= -\left(\frac{\Delta - \|s(y)\|_2}{\Delta \ \|s(y)\|_2}\right)\left(-\frac{\|s(y)\|_2^3}{s(y)^T \nabla_y s(y)}\right) \\ &= \left(\frac{\|s(y)\|_2 - \Delta}{\Delta}\right)\frac{\|s(y)\|_2^2}{\|w(y)\|_2^2}. \end{aligned}$$

Newton's method for (3.36) is given in Algorithm 3.40.

When $\lambda_1 < 0$ and there is no value of $y \in (-\lambda_1, \infty)$ such that $\|s(y)\|_2 = \Delta$, the situation is known as the *hard* case. The hard case arises when g is orthogonal to the eigenvector u_1 corresponding to the eigenvalue λ_1 with $\lambda_1 < 0$, i.e. $g^T u_1 = 0$ and Δ is too big. In the hard case $y = -\lambda_1$ and $B + yI$ is a singular (but consistent) system—it is consistent precisely because $g^T u_1 = 0$. We compute the *critical step* s_{cri} by the following limit

$$s_{cri} = \lim_{y \to -\lambda_1} s(y) = \sum_{i=2}^{n} \frac{\gamma_i}{(\lambda_i + y)}, \tag{3.37}$$

Since u_1 is a eigenvector corresponding to eigenvalue λ_1, $(B - \lambda_1 I)u_1 = 0$. So this system

has other solutions $s + \tau u_1$ for any τ, because

$$(B + yI)(s_{cri} + \tau u_1) = -g.$$

Using the trust-region constraint we find that

$$\|s_{cri} + \tau u_1\|_2^2 = \Delta^2, \tag{3.38}$$

which is a quadratic equation for the unknown τ, and either root suffices.

The Algorithm 3.40 is globally and ultimately quadratically convergent when started in the interval $[-\lambda_1, y^*]$ except in the hard case, but needs to be safeguarded to make it robust for the hard cases. Notice that the main computational cost per iteration is a Cholesky decomposition of $B + yI$.

Algorithm 3.40 ***Exact trust-region [10]***
Let $y > -\lambda_1$ *and* $\Delta > 0$ *be given.*
Until "convergence" do:
Factorize $B + yI = LL^T$.
Solve $LL^T s = -g$ *for* s.
Solve $Lw = s$ *for* w.
Replace y *by*

$$y + \left(\frac{\|s\|_2 - \Delta}{\Delta}\right)\left(\frac{\|s\|_2^2}{\|w\|_2^2}\right).$$

Chapter 4

Nonlinear system of equations and nonlinear least squares

4.1 Introduction

We consider the nonlinear system of equations of the form

$$F(x) = 0, \tag{4.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable and $x \in \mathbb{R}^n$. F will be called the *nonlinear residual*, or simply the *residual*. The Jacobian matrix $A(x)$ of $F(x)$ is given by

$$A(x) = [\nabla F_1(x), \nabla F_2(x), \cdots, \nabla F_n(x)]^T, \tag{4.2}$$

where $F_1(x), F_2(x), \cdots$ are the components of $F(x)$.

Iterative methods must be used to solve the nonlinear equations (4.1) since closed-form solutions can be rarely obtained. Newton's method is the most widely used iterative method in applications. In Newton's method, for a given initial point $x_0 \in \mathbb{R}^n$, a sequence of iterates $x_k, k = 1, 2, 3, \cdots$ is generated in such a way that hopefully the approximation to some solution is progressively improved. Iterates $x_k, k = 0, 1, 2, \cdots$ are computed as follows.

$$\text{solve } A(x_k)s_k = -F(x_k) \quad \text{for} \quad s_k \tag{4.3a}$$

$$x_{k+1} = x_k + s_k \ \{ \text{ update the solution } \}, \tag{4.3b}$$

where s_k is the *Newton step*. The algorithm described by (4.3) is known as Newton's method or *local Newton method*. The Newton iteration is well-defined whenever $A(x_k)$ is nonsingular.

A nonlinear system of equations may have a single solution, more than one solution, or even no solution. Convergence of any method to a solution can be guaranteed if the solution exists. Newton's method converges rapidly to a solution if the initial point starts

close enough to the solution. The rate of convergence is usually quadratic. Newton's method requires substantial amount of work per iteration, for a problem with dense Jacobian matrix, each iteration requires $O(n^2)$ scalar function evaluations to form the Jacobian matrix and $O(n^3)$ arithmetic operations to solve the linear system (4.3a) [17, 21, 25, 28, 31].

Now we show the fast local convergence property of Newton's method provided that the initial point is close enough to a solution. The method may not converge at all from a poor initial point. Therefore, we need to incorporate line-search/trust-region methods to achieve convergence which will be discussed later.

Theorem 4.1 ***Fast local convergence of Newton's method [40]***
Suppose that $F : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable and $x^* \in \mathbb{R}^n$ is a solution of $F(x) = 0$. If $A(x^*)$ is nonsingular, then there exists an $\epsilon > 0$ such that

1. *$A(x)$ is nonsingular with $\|A(x)^{-1}\| \leq 2\|A(x^*)^{-1}\| \quad \forall x \in \mathbb{B}_\epsilon(x^*)$.*
2. *x^* is the one and only one solution of $F(x) = 0$ for all $x \in \mathbb{B}_\epsilon(x^*)$, and the following condition holds:*

$$\frac{1}{2\|A(x^*)^{-1}\|}\|x - x^*\| \leq \|F(x)\| \leq 2\|A(x^*)\|\|x - x^*\|.$$

3. *For all $x_0 \in \mathbb{B}_\epsilon(x^*)$, the algorithm given by (4.3) generates a sequence $\{x_k\} \subset \mathbb{B}_\epsilon(x^*)$ which converges Q-superlinearly to x^*.*
4. *If A is Lipschitz continuous in $\mathbb{B}_\epsilon(x^*)$ with a constant L, then $\{x_k\}$ converges Q-quadratically to x^*, i.e.,*

$$\|x_{k+1} - x^*\| \leq L\|A(x^*)^{-1}\|\|x_k - x^*\|^2 \quad \forall k \geq 0.$$

4.2 Global convergence

Local Newton method necessitates the initial point to be near the solution. Otherwise, the method may fail to converge. There are many forms of failure: the method may break down, the iterates may oscillate between two non-stationary points, or they can converge to a non-stationary point [9]. We now discuss Newton's method with global convergence properties where the initial point may lie anywhere and the method converges to a solution. The global convergence property of Newton's method for a nonlinear system of equations can be obtained using the Armijo line search (section 3.4.1). By introducing a scalar α_k in (4.3b), we have that

$$x_{k+1} = x_k + \alpha_k s_k, \tag{4.4}$$

where $0 < \alpha_k \leq 1$ is the *step length* and is chosen based on sufficient reduction of a merit function. When we discuss global convergence, s_k will be called the *Newton search direction* and $\alpha_k s_k$ will be called the Newton step. Note that the local Newton method uses full steps ($\alpha_k = 1$). We consider the following well-known merit function [9].

$$\phi(x) = \frac{1}{2}\|F(x)\|_2^2. \tag{4.5}$$

Sufficient reduction of $\phi(x)$ can be checked by the Armijo rule

$$\phi(x_k + \alpha_k s_k) - \phi(x_k) \leq \gamma \alpha_k \nabla\phi(x_k)^T s_k \tag{4.6}$$

$$\Rightarrow \quad \underbrace{F(x_k + \alpha_k s_k)^T F(x_k + \alpha_k s_k) - F(x_k)^T F(x_k)}_{ared} \leq \gamma \underbrace{\alpha_k A(x_k)^T F(x_k) s_k}_{pred}. \tag{4.7}$$

Now we give the following algorithm of the Armijo line search with modification from Algorithm 3.7.

Algorithm 4.2 ***Armijo line search***

Given s_k *and* x_k;
$\alpha_k = 1$;
$for \quad l = 0, 1, 2, \cdots$
    $x_{k+1} = x_k + \alpha_k s_k$
    Stop, if the Armijo rule (4.7) is satisfied
    $\alpha_k = \dfrac{\alpha_k}{2}$;
$end(for)$

In most cases, Newton's method with line search makes steady progress towards a solution. But the method can still fail if Jacobian matrices are singular. We now examine this issue.

4.3 Singularity issue

As we have discussed before, Newton's method is well-defined if the Jacobian A is non-singular. In case of a singular A, we may be unable to solve the linear system (4.3a) by efficient numerical methods like LU decomposition [12]. However there are some more expensive but robust numerical methods, e.g., singular value decomposition (SVD) [33], QR decomposition [12] which can be used to avoid this deficiency. Using SVD for A, we obtain

$$UDV^T = A,$$

where U is an $n \times n$ orthonormal matrix, D an $n \times n$ diagonal matrix, and V an $n \times n$ orthonormal matrix. D contains the *singular values* d_j at position (j, j) with $j =$

$1, 2, \cdots, n$. If A is nonsingular, i.e., $d_j \neq 0$ for $j = 1, 2, \cdots, n$, the solution can be obtained by

$$s_k = -VD^{-1}(U^T F(x_k)),$$

where D^{-1} is a diagonal matrix with elements $(1/d_j)$ at (j, j). If singularity occurs, then at least one of $d_j,\ j = 1, 2, \cdots, n$ is zero. To avoid singularity, a small number μ is added to d_j when $d_j = 0$. This strategy is called *regularization*. In the following regularization algorithm, we replace d_j by μ when $d_j < \mu$.

Algorithm 4.3 ***Solving linear system by SVD [9]***

$UDV^T = A(x_k);$
$for \quad i = 1, 2, \cdots$
$\quad if \quad (d_j < \mu)$
$\quad\quad d_j = \mu;$
$end(for)$
$s_k = -V\ diag([1/d_1, 1/d_2 \cdots 1/d_n])\ (U^T F(x_k))$

Algorithm 4.3 does not perform well as the chosen value of μ may not be optimal in computing the next iterate x_{k+1}. Now we attempt to find an optimal μ using the trust-region method.

4.4 Trust-region method

Problem (4.3a) is a particular case of the minimization problem:

$$\min_s \quad \frac{1}{2}\|As + F\|_2^2 \quad = \quad \min_s \quad \underbrace{\frac{1}{2}F^T F}_{f} + s^T \underbrace{A^T F}_{g} + \frac{1}{2} s^T \underbrace{A^T A}_{B} s$$

For brevity, we have omitted the iteration subscript k. The matrix $A^T A$ is symmetric positive semi-definite. By applying the trust-region constraint to the above problem we obtain

$$\min_s \quad \underbrace{\frac{1}{2}F^T F}_{f} + s^T \underbrace{A^T F}_{g} + \frac{1}{2} s^T \underbrace{A^T A}_{B} s \quad \text{subject to } \|s\|_2 \leq \Delta, \tag{4.8}$$

where Δ is a trust-region radius. Using first-order necessary condition, the above problem reduces to

$$\text{Solve } (A^T A)s = -A^T F \text{ for } s \quad \text{with constraint } \ \|s\|_2 \leq \Delta. \tag{4.9}$$

The system $(A^TA)s = -A^TF$ is known as a system of *normal equations*. Now we can solve the trust-region subproblem (4.8) using Theorem 3.39:

$$(A^TA + yI)s^* = -A^TF, \tag{4.10a}$$

$$y(\Delta - \|s^*\|_2) = 0, \tag{4.10b}$$

$$(A^TA + yI) \text{ is positive semidefinite}, \tag{4.10c}$$

where $y \geq 0$ is a Lagrange multiplier.

In case of nonsingular A, conventional methods are far more efficient than trust-region methods, especially when n is large. For a large n, rounding error will increase and may have influence on the accuracy of the solution. The condition number of A^TA is much worse than that of A. The 2-norm condition number for A^TA is exactly the square of the condition number of A which could cause difficulties. Any progress made in one step of the iterative procedure may be annihilated by the numerical errors. On the other hand, if the original matrix has a good 2-norm condition number, then the normal equation approach should not cause any serious difficulties [33].

Moreover, the minimization problem $\min_s \frac{1}{2}\|As + F\|_2$ has a much better condition number than the linear system $(A^TA)s = -A^TF$ [13, p. 73]. Hence, we want to avoid building A^TA and A^TF.

We now present a practical technique which uses the original form (4.3a) for the solution s avoiding the multiplications A^TA and A^TF. We will make a reform of (4.10a). First, we apply the QR decomposition on A:

$$QR = A,$$

where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix, and $R \in \mathbb{R}^{n\times n}$ is an upper triangular matrix. We can use Householder transformations [17] to compute the QR decomposition efficiently. Now (4.10a) gives us

$$(A^TA + yI)s = -A^TF \tag{4.11a}$$

$$\Rightarrow \quad (R^TQ^TQR + yI)s = -R^T\underbrace{Q^TF}_{b} \tag{4.11b}$$

$$\Rightarrow \quad (R^TR + yI)s = -R^Tb \tag{4.11c}$$

$$\Rightarrow \quad \begin{pmatrix} R^T & \sqrt{y}I \end{pmatrix} \begin{pmatrix} R \\ \sqrt{y}I \end{pmatrix} s = -\begin{pmatrix} R^T & \sqrt{y}I \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{4.11d}$$

The above equation can be rewritten as

$$\underbrace{\begin{pmatrix} R \\ \sqrt{y}I \end{pmatrix}}_{Q_1R_1} s = -\begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{4.12}$$

where $Q_1 \in \mathbb{R}^{2n\times n}$ and $R_1 \in \mathbb{R}^{n\times n}$ can be computed by the QR decomposition again. We

can save some computational time using Givens rotations [17] for this QR decomposition. We need to eliminate the diagonal elements of the lower part $\sqrt{y}I$ of this system matrix. But while doing so, we will get some additional fill-ins which have to be eliminated too. This algorithm is given below.

Algorithm 4.4 ***Givens rotation for the system (4.12) [31]***
for $i = n, n-1, \cdots, 1$

- *j = 0;*
- *To eliminate* (i,i) *of* $\sqrt{y}I$*, rotate row* i *of* R *with row* i *of* $\sqrt{y}I$*.* $\{$*Row* $(n-j)$ *of the matrix* $\sqrt{y}I$ *receives fill-ins at columns* $n-j+1$ *to* n *which will be eliminated in the following loop over* k*.*$\}$
- *for* $k = n-j+1, \cdots, n$
  - *Rotate row* k *of* R *with row* $n-j$ *of* $\sqrt{y}I$ *to eliminate fill-in at* $(n-j,k)$ *of* $\sqrt{y}I$*.*
- *end(for)*
- *j = j+1;*

end(for).

The value y can be updated iteratively by a similar idea described in section 3.9.3. The update formula of y_p is given by

$$y_{p+1} = y_p + \left(\frac{\|s\|_2^2}{\|w\|_2^2}\right)\frac{\|s\|_2 - \Delta}{\Delta}.$$

Setting $\mu = \sqrt{y}$ we deduce that

$$\mu_{p+1} = \sqrt{\mu_p^2 + \left(\frac{\|s\|_2^2}{\|w\|_2^2}\right)\frac{\|s\|_2 - \Delta}{\Delta}}, \tag{4.13}$$

where

$$\begin{aligned} \|w\|_2^2 &= s^T(R^TR + y_pI)^{-1}s \\ &= s^T\left[\begin{pmatrix} R & \mu_pI \end{pmatrix}\underbrace{\begin{pmatrix} R \\ \mu_pI \end{pmatrix}}_{Q_1R_1}\right]^{-1} s. \end{aligned}$$

Hence we derive

$$\begin{aligned} \|w\|_2^2 &= s^T[(Q_1R_1)^TQ_1R_1]^{-1}s \\ &= s^T[R_1^TR_1]^{-1}s \\ &= (R_1^{-T}s)^TR_1^{-1}s = \|R_1^{-T}s\|_2^2. \end{aligned}$$

Similar to the trust-region method for unconstrained optimization given in Algorithm 3.40, we present the trust-region method for nonlinear equations in the following algorithm.

Algorithm 4.5 ***Trust-region method for nonlinear equations***

Given $\Delta > 0, \kappa_s > 0, R, b, \mu_0 = 0$*;*

for $p = 0, 1, 2, \cdots$

Solve $\underbrace{\begin{pmatrix} R \\ \mu_p I \end{pmatrix}}_{Q_1R_1} s = -\begin{pmatrix} b \\ 0 \end{pmatrix}$ *for* s;

if $\|s\|_2 \leq \Delta$ *and* $\mu_p = 0$, *then stop with the solution* s;

if $|(\|s\|_2 - \Delta)| \leq \kappa_s\Delta$, *then stop with the solution* s;

Update μ *by*

$$\mu_{p+1} = \sqrt{\mu_p^2 + \left(\frac{\|s\|_2^2}{\|R_1^{-T}s\|_2^2}\right)\frac{\|s\|_2 - \Delta}{\Delta}};$$

end(for)

The common chosen value of κ_s is 0.1. When R is well-conditioned, the algorithm will immediately stop with the solution s at $p = 0$. In case of an ill-conditioned (or singular) R, the algorithm iteratively computes s by optimizing μ such that $\|s\|_2 \approx \Delta$.

4.5 Termination criteria

The error at the kth iteration is given by

$$e = x_k - x^*.$$

So we do not know the error without knowing the solution as the iteration progresses. But we can use $F(x)$ as an indicator of rate of decay in $\|e\|$. Hence, we have the termination criteria [25]

$$\|F(x)\| \leq \tau_r\|F(x_0)\| + \tau_a. \tag{4.14}$$

The *relative error tolerance* τ_r and the *absolute error tolerance* τ_a are both significant in the termination. An initial point near the solution may make (4.14) impossible to satisfy if we use only the relative error tolerance (by setting $\tau_a = 0$).
When the difference between successive iterates is small, the approximate solutions are not changing much and (hopefully!) the iterate has converged. This gives an another stopping criteria [21]

$$\frac{\|x_{k+1} - x_k\|}{\max(\|x_{k+1}\|, \|\hat{x}\|)} \leq \tau, \tag{4.15}$$

where $\hat{x}$ is a typical x and is commonly taken as the initial point. If the trust-region radius is very small, we have a very small progress in further iterations. In this case, it is reasonable to stop the iterations.

4.6 Complete algorithm

Now we combine the previous considerations and algorithms. The resulting algorithm can be stated as follows.

Algorithm 4.6 ***Newton's method with trust-region and Armijo line search***
Choose an initial point x_0.
for $k = 0, 1, 2, \cdots$

- *Compute the residual* $F(x_k)$ *and Jacobian matrix* $A(x_k)$.
- *Apply Algorithm 4.5 in order to compute the search direction* s_k.
- *Apply Algorithm 4.2 in order to compute the next iterate* x_{k+1}.
- *Stop if convergence criteria is satisfied.*

end(for)

4.7 Nonlinear least squares method

Given input-output pairs $(t_i, y_i), i = 1, .., m$, we want to find a vector $x \in \mathbb{R}^n$ that gives the best fit in the least square sense to model a function $h(t, x)$, where $h : \mathbb{R}^n \to \mathbb{R}^m$.

We define components of the *residual* $F : \mathbb{R}^n \to \mathbb{R}^m$ by

$$F_i(x) = y_i - h(t_i, x) \quad \forall i = 1, .., m.$$

The aim is to compute the vector x which minimizes the difference between y_i and $h(t_i, x)$ for all $i = 1, .., m$. This can be considered as a problem where the sum of squares of residual components is to be minimized, and hence we obtain a minimization problem of

the form

$$\min_x \phi(x) = \min_x \ \frac{1}{2} F(x)^T F(x). \tag{4.16}$$

We take the value $\frac{1}{2}$ for convenience and it has no effect on the optimal value of x. In practice, m is significantly larger than n and the experimental errors yield $y_i \neq h(t_i, x)$ even with a best possible x [10].

Applying Newton's method to (4.16), a Newton step s_k for the current iterate x_k can be obtained from

$$\text{solve} \ \underbrace{\nabla^2 \phi(x_k)}_{H(x_k)} s_k \quad = \quad -\underbrace{\nabla \phi(x_k)}_{g(x_k)} \quad \text{for} \quad s_k. \tag{4.17}$$

The gradient vector $g(x_k)$ and the Hessian matrix $H(x_k)$ are given by

$$g(x_k) \quad = \quad \sum_{i=0}^{m} \nabla F_i(x_k) F_i(x) = \underbrace{\nabla F(x_k)}_{A(x_k)}{}^T F(x_k) = A(x_k)^T F(x_k) \tag{4.18a}$$

$$H(x_k) \quad = \quad A(x_k)^T A(x_k) + \underbrace{\sum_{i=0}^{m} \nabla^2 F_i(x_k) F_i(x)}_{S_k} = A(x_k)^T A(x_k) + S_k. \tag{4.18b}$$

The later term S_k of (4.18b) is usually inconvenient and expensive to compute. To avoid much computing cost, this term is omitted and we have

$$H(x_k) \approx A(x_k)^T A(x_k). \tag{4.19}$$

Hence (4.17) can be rewritten as

$$A(x_k)^T A(x_k) s_k = -A(x_k)^T F(x_k),$$

which is the system of normal equations for the *Gauss-Newton equations*

$$A(x_k) s_k = -F(x_k). \tag{4.20}$$

To avoid singularity in $A(x_k)$, we use the regularization (described in section 4.3) for (4.20) which gives the *Levenberg-Marquardt equations*

$$\begin{pmatrix} A(x_k) \\ \sqrt{y_k} I \end{pmatrix} s = - \begin{pmatrix} F(x_k) \\ 0 \end{pmatrix}, \tag{4.21}$$

where $\sqrt{y_k}$ is a nonnegative parameter chosen by some strategy in order to avoid ill-condition or rank deficiency of the linear least squares sub-problem (4.20).

To get a trust-region-based solution, we can use Algorithm 4.6 for the Gauss-Newton equations (4.20) directly. If $\|F(x)\|$ is very large (the so-called large residuals problem), the Gauss-Newton equations are not a good approximation and the convergence may not

be guaranteed. In such a case, we require a better approximation of the Hessian given in (4.18b). We could use the trust-region method of unconstrained optimization to (4.16) directly using a finite difference (see section 5.4) for Hessian of $\nabla^2\phi(x_k)$. Another idea is to use quasi-Newton approximation for the expensive part S_k of (4.18b). Dennis, Gay, and Welsch derived the following update formula [21, chap. 10], [31, chap. 10] for S_k.

$$S_{k+1} = S_k + \frac{(z - S_k d)w^T + w(z - S_k d)^T}{w^T d} - \frac{(z - S_k d)^T d}{(w^T d)^2} w w^T, \tag{4.22}$$

where

$$\begin{aligned} d &= x_{k+1} - x_k \\ w &= A(x_{k+1})^T F(x_{k+1}) - A(x_k)^T F(x_k) \\ z &= A(x_{k+1})^T F(x_{k+1}) - A(x_k)^T F(x_{k+1}). \end{aligned}$$

However the matrix S_k is not guaranteed to vanish as the iterates approach to a zero-residual solution. This deficiency can be avoided by scaling S_k prior to an update. We substitute S_k by $\tau_k S_k$ on the right-hand-side of (4.22) where

$$\tau_k = \min\left(1, \frac{|d^T z|}{|d^T S_k d|}\right).$$

We observe that the part $A(x_k)^T A(x_k)$ of $\nabla^2\phi(x_k)$ can be readily available, as $A(x_k)$ can be computed analytically or by finite differences. So it might be reasonable to consider a hybrid algorithm, which will try a Gauss-Newton approach if the residual is small (or it produces a sufficiently good step), but switch to a quasi-Newton approach if otherwise [17, 20, 31].

Chapter 5

Implementation of algorithms

5.1 Unconstrained optimization

Algorithm 3.30 of Newton's method with a trust-region technique was implemented for solving the nonlinear unconstrained optimization problems. The flow chart of this algorithm is given by FIGURE 5.1. Note that we have Newton's method with line search if the trust-region technique is discarded from the figure. Moreover, we obtain local Newton method by removing the trust-region and Armijo line search techniques. So it suffices to consider the algorithm of Newton's method with the trust-region technique. This algorithm uses several methods and considerations from Chapter 3 and is organized similar to algorithms described in [7, chap. 6]. We use reasonable constant values considering the accuracy of the solution and the expense of computation.

1. Set constants: $\epsilon = 1.0e\text{-}6$, $\Delta = 1.0$.
2. Initialize the point x.
3. Evaluate the function $f(x)$ and the gradient $g(x)$ (see section 5.4).
4. Stop with x as a solution, if one of the conditions
   (a) $\|g(x)\| \leq \tau_r \|g(x_0)\| + \tau_a$ with $\tau_r = 1.0e\text{-}6$, $\tau_a = 1.0e\text{-}6$.
   (b) $\frac{\|s\|}{\max(\|x\|, \|\hat{x}\|)} \leq \tau$, where $\tau = 1.0e\text{-}6$, $\hat{x}$ has the value of x at the first iteration and and s is the current search direction.
   (c) $\Delta \leq \epsilon$.

   is satisfied.
5. Evaluate the Hessian $H(x)$ (see section 5.4).
6. Compute the search direction s using the trust-region method described in section 5.1.1.
7. Compute the step length α using Armijo line search. The value of x is updated if the Armijo condition is satisfied. The value of Δ is modified if necessary. This will be described in section 5.1.2
8. Go to step 2.

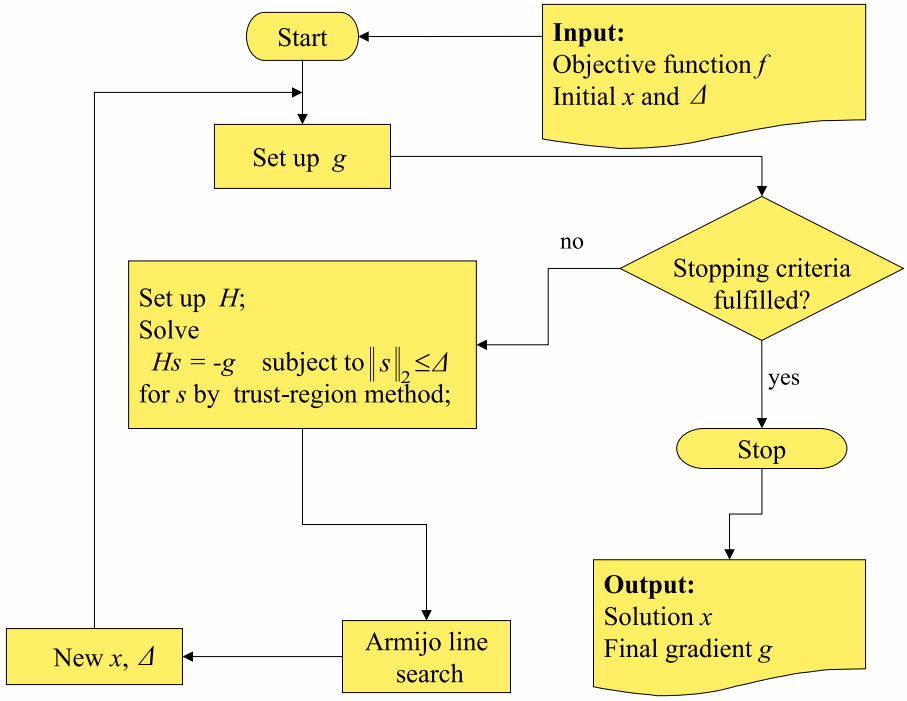

FIGURE 5.1 Flow chart of the main program

5.1.1 Trust-region method

Algorithm 3.40 was implemented to solve the trust-region subproblems of the unconstrained optimization. This algorithm is arranged similar to algorithms described in [10, p. 193–199]. The flow chart of the algorithm is given by FIGURE 5.2. In this algorithm, the system (3.33) is solved for an initial Lagrange multiplier $y = 0$. It stops when the condition $\|s\|_2 \leq \Delta$ is satisfied. If this condition is not satisfied, a new y is computed and the condition $\|s\|_2 \approx \Delta$ is checked after solving (3.34). We may require to go through this process several times before $\|s\|_2$ will come close to Δ. We set the constants $\kappa_s = 0.1$, $maxiter = 4$ to avoid much expense in the computation. Here we give the detail description.

Input:

system matrix M, system vector g, trust-region radius Δ, tolerance for simple case κ_s, and constant θ.

1. Set the constants $\theta = 0.1, \kappa_s = 0.1$. Set $maxiter = 4, iter = 0$.

2. Compute the upper bound y_u of the Lagrange multiplier y by the ∞-norm:

$$\begin{aligned} y_u &= \|M\|_\infty \\ &= \max_i \left(|M_{ii}| + \sum_{j \neq i} |M_{ij}| \right). \end{aligned}$$

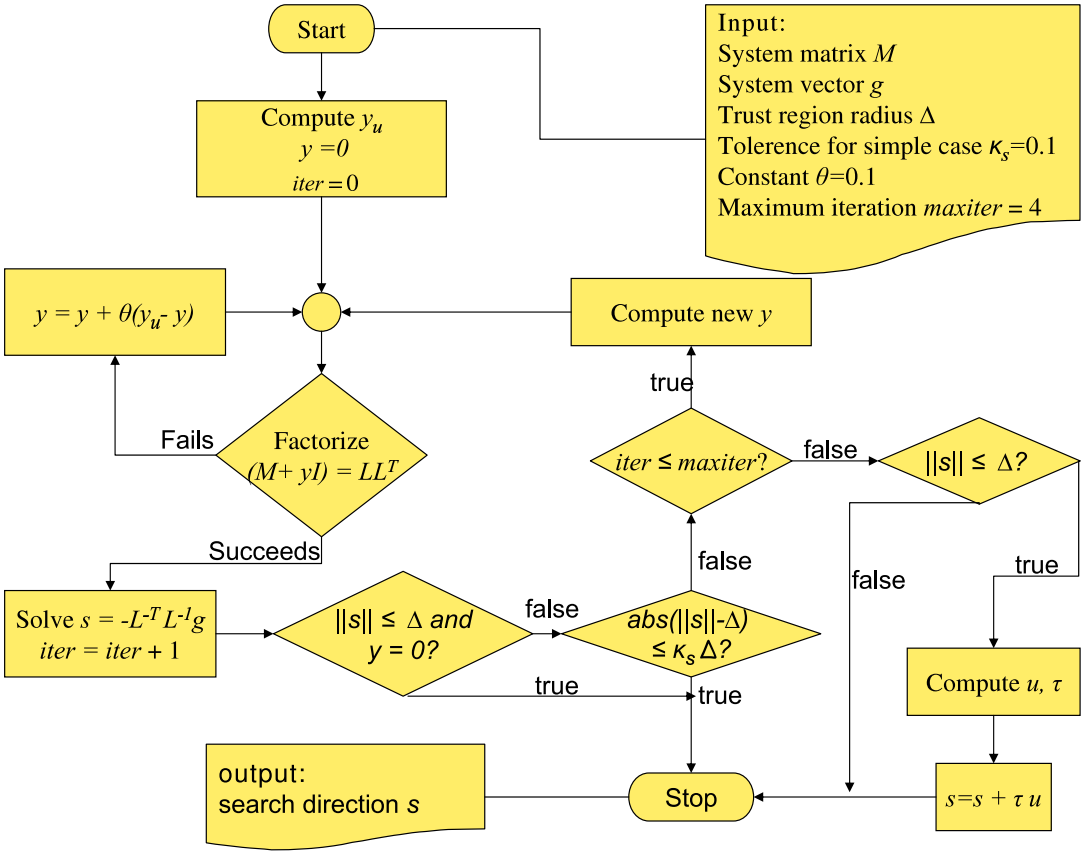

FIGURE 5.2 Flow chart of the trust-region method

Set $y = 0$.

3. Decompose $(M + yI) = LL^T$ by the Cholesky factorization (see section 2.8.1). Here L is a lower triangular matrix with the same dimension of M.

4. If the decomposition is not successful, then

   (a) Set $y = y + \theta(y_u - y)$.

   (b) Go to step 3.

5. Solve $LL^T s = -g$ for s and set $iter = iter + 1$.

   /* convex case */

6. If $y = 0$ and $\|s\|_2 \leq \Delta$, then stop with the value s.

   /* simple case */

7. If $|(\|s\|_2 - \Delta)| \leq \kappa_s \Delta$, then stop with the value s.

8. Compute new y with the formula

$$\text{Solve } Lw = s \text{ for } w,$$
$$y = y + \left(\frac{\|s\|_2^2}{\|w\|_2^2}\right)\frac{\|s\|_2 - \Delta}{\Delta},$$

   where w is a vector with the same dimension of s.

9. If $iter \leq maxiter$, then go to step 3.

10. If $\|s\|_2 < \Delta$, then /* hard nonconvex case */

    (a) Compute approximate eigenvector u (Section 5.3). Determine τ from (3.38).

    (b) Set $s = s + \tau u_1$.

Output:
search direction s.

We notice that well-conditioned problems are solved with a few iterations. In case of an ill-conditioned system matrix M, more iterations are required to make M well-conditioned by adding a new Lagrange multiplier y at the diagonals of M in each iteration. A new y is computed in such a way that the solution satisfies the trust-region constraint $\|s\|_2 \approx \Delta$.

5.1.2 Armijo line search

The algorithm of Armijo line search is described by FIGURE 5.3. We determine a step length which reduces the merit function sufficiently. For this purpose, we compute the predicted and actual reductions with the initial step length α. If the reduction ratio ρ (ratio of actual to predicted reduction) is more than a certain threshold γ_1, we allow the step. Otherwise we reduce α in an iterative way as long as $\rho \leq \gamma_1$. We set the maximum number of these iterations to 4. If the reduction is sufficient, we increase the trust-region radius Δ. Otherwise we reduce Δ. We now present the detail description.
Input:
constants $\Delta_{max} = 1000, \gamma_1 = 0.001, \gamma_2 = 0.1, \gamma_3 = 0.75$, current iterate x, search direction s and trust-region radius Δ.

1. Set the step length $\alpha = 1$.

2. Compute the predicted reduction *pred* from the right hand side of (3.4).

3. for $iter = 0, 1, 2, 3$

   (a) Compute the current actual reduction *ared* from the left hand side of (3.4).

   (b) Compute the reduction ratio $\rho := \frac{ared}{pred}$.

   (c) If $(\rho > \gamma_1)$, then perform the following operations.

      i. Set $x = x + \alpha s$.

      ii. If $(\rho > \gamma_3)$, then set $\Delta = \min(2\Delta, \Delta_{max})$.

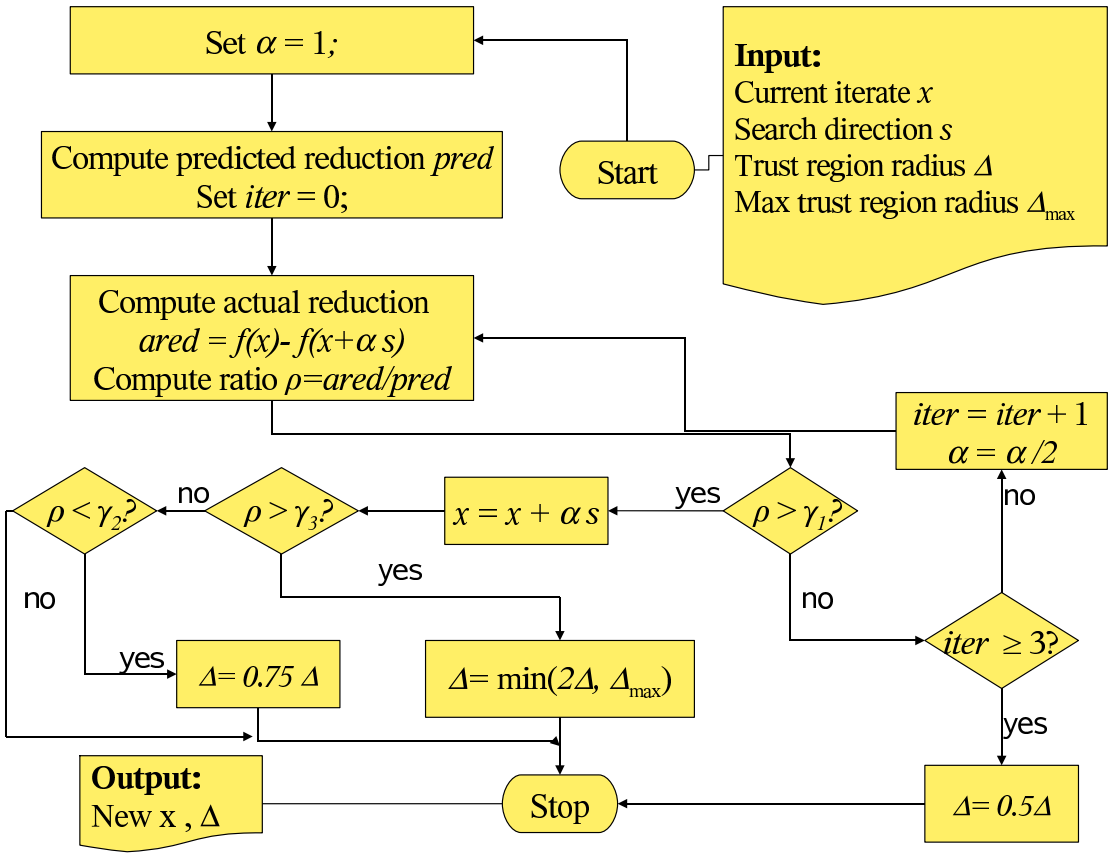

FIGURE 5.3 Flow chart of Armijo line search

iii. If $(\rho < \gamma_2)$, then set $\Delta = 0.75\Delta$.

iv. Stop with the new values of x and Δ.

(d) set $\alpha = \frac{\alpha}{2}$.

4. If $(\rho \leq \gamma_1)$, then set $\Delta = 0.5\Delta$.

Output:
updated x and Δ.

5.2 Nonlinear equations and nonlinear least squares

Algorithm 4.6 was implemented to solve the nonlinear equations (nonlinear least squares) by Newton's (Gauss-Newton) method with a trust-region technique. The flow chart of the algorithm is given by FIGURE 5.4. Note that we have Newton's (Gauss-Newton) method with line search if the trust-region technique is discarded from the figure. Moreover, we obtain local Newton method (Gauss-Newton) by removing the trust-region and Armijo line search techniques. So it suffices to consider Newton's (Gauss-Newton) method with the trust-region technique which we illustrate now. This algorithm draws together the discussion of Chapter 4 and is structured similar to the algorithm described in Section

5.1. We use reasonable values for constants considering the accuracy of the solution and the expense of computation.

1. Set constants: $\epsilon = 1.0e\text{-}6$, $\Delta = 1.0$.
2. Initialize the point x.
3. Compute the residual $F(x)$.
4. Stop with x as a solution, if any of the conditions
   (a) $\|F(x)\| \leq \tau_r \|F(x_0)\| + \tau_a$ with $\tau_r = 1.0e\text{-}6$, $\tau_a = 1.0e\text{-}6$.
   (b) $\frac{\|s\|}{\max(\|x\|, \|\hat{x}\|)} \leq \tau$, where $\tau = 1.0e\text{-}6$, $\hat{x}$ has the value of x at the first iteration and s is the current search direction.
   (c) $\Delta \leq \epsilon$.

   is satisfied.
5. Compute the Jacobian $A(x)$ of $F(x)$ (similar procedure described in section 5.4).
6. Perform QR decomposition of A. Compute $b = Q^T F$.
7. Compute the search direction s using the trust-region method on (4.12) which will be discussed later.
8. Compute the step length α with the Armijo line search. The value of x is updated if the Armijo condition is satisfied. The value of Δ is modified if necessary. We will discuss these later.
9. Go to step 2.

5.2.1 Trust-region method

Algorithm 4.5 was implemented to solve the trust-region subproblems of the nonlinear equations, or nonlinear least squares problems. The flow chart of the algorithm is given by FIGURE 5.5. The implementation procedure is almost similar to the procedure for the unconstrained optimization discussed in section 5.1.1. We now give the detail description of trust-region algorithm.

Input:

constant ϵ, system matrix R, system vector b, trust-region radius Δ, tolerance for simple case κ_s.

1. Set the constants $\epsilon = 1.0e\text{-}6$, $\kappa_s = 0.1$. Set $maxiter = 4, iter = 0$.
2. Set $\mu = \epsilon$ if R is singular. Otherwise, set $\mu = 0$.
3. Solve (4.12) for s. Set $iter = iter + 1$.
4. If $\mu = 0$ and $\|s\|_2 \leq \Delta$, then stop with the value s.

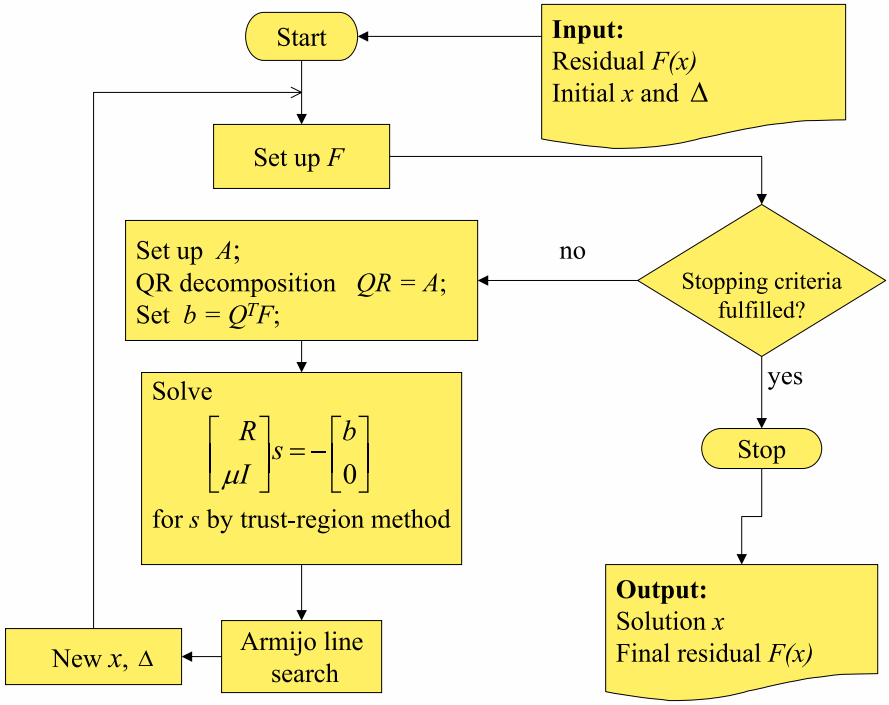

FIGURE 5.4 Flow chart of the main program

5. If $|(\|s\|_2 - \Delta)| \leq \kappa_s \Delta$, then stop with the value s.

6. Update μ by (4.13).

7. If $iter \leq maxiter$, then go to step 3.

Output:
search direction s.

5.2.2 Armijo line search

The algorithm of Armijo line search is described by FIGURE 5.6. The implementation procedure is almost the same as the procedure for the unconstrained optimization discussed in section 5.1.2.

Input:
constants $\Delta_{max} = 1000, \gamma_1 = 0.001, \gamma_2 = 0.1, \gamma_3 = 0.75$, current iterate x, search direction s, residual F, Jacobian matrix A and trust-region radius Δ.

1. Set the step length $\alpha = 1$.

2. Compute the predicted reduction *pred* from the right hand side of (4.7).

3. for $iter = 0, 1, 2, 3$

   (a) Compute the current actual reduction *ared* from the left hand side of (4.7).

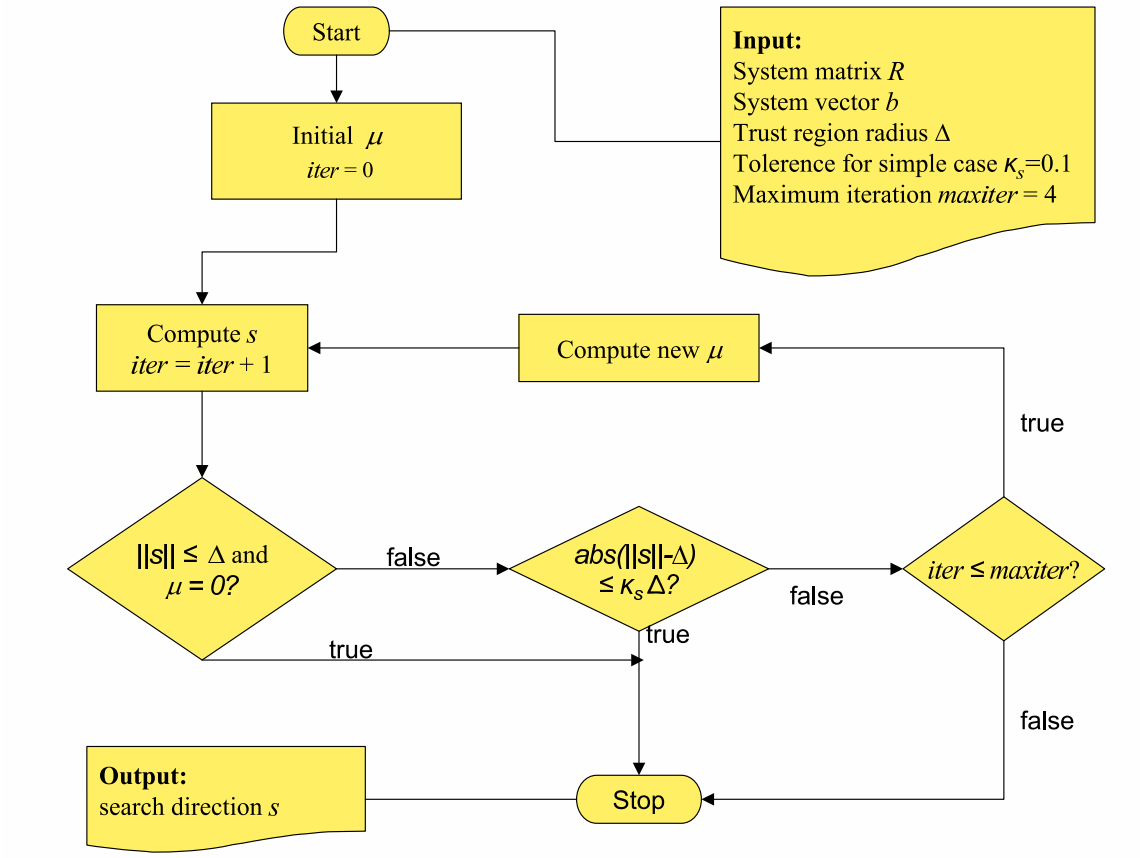

FIGURE 5.5 Flow chart of trust-region method

(b) Compute the reduction ratio $\rho := \frac{ared}{pred}$.

(c) If $(\rho > \gamma_1)$, then perform the following operations.

i. Set $x = x + \alpha s$.

ii. If $(\rho > \gamma_3)$, then set $\Delta = \min(2\Delta, \Delta_{max})$.

iii. If $(\rho < \gamma_2)$, then set $\Delta = 0.75\Delta$.

iv. Stop with the new values of x and Δ.

(d) set $\alpha = \dfrac{\alpha}{2}$.

4. If $(\rho \leq \gamma_1)$, then set $\Delta = 0.5\Delta$.

Output:
updated x and Δ.

5.3 Computing u of the trust-region method

We need the unit eigenvector u corresponding to the leftmost eigenvalue λ_1 of a symmetric matrix M in the hard case of the trust-region subproblem (see section 3.9.3). Since u is

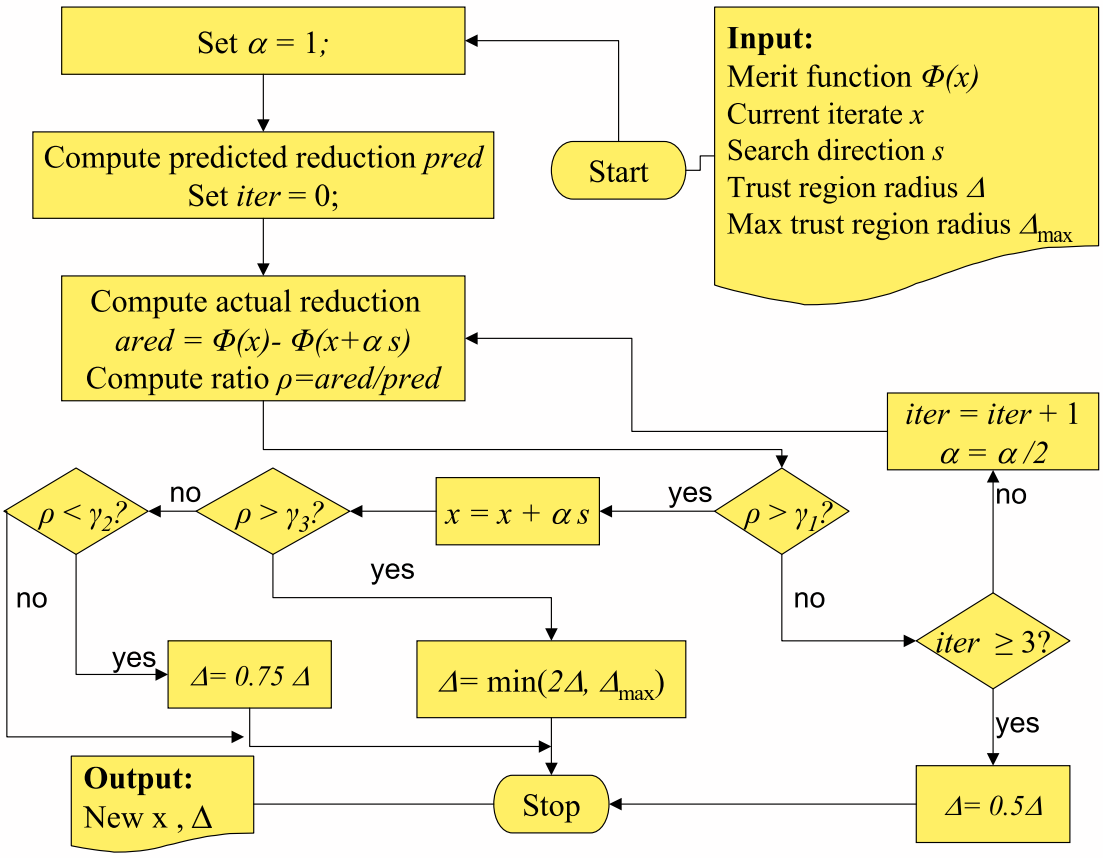

FIGURE 5.6 Flow chart of Armijo line search

the eigenvector corresponding to λ_1, we can write

$$(M - \lambda_1 I)u = 0.$$

Now we describe how the vector u can be computed in a cheap way [7, p. 58]. We replace $-\lambda_1$ by its close value y. The solution u of the above equation is the minimum of $u^T(M+yI)u$. This is equivalent to compute a vector v which maximizes $v^T(M+yI)^{-1}v$. So we have

$$v^T \left(\underbrace{M + yI}_{LL^T} \right)^{-1} v = v^T L^{-T} \underbrace{L^{-1}v}_{w} = w^T w.$$

We will choose the component of v as ± 1 in order to make w large. This is achieved by choosing the signs of entries of v at the stage of forward substitution $w = L^{-1}v$. Specifically, at the kth component we have

$$L_{kk} w_k = v_k - \sum_{i=1}^{k-1} L_{ki} w_i.$$

We take v_k to be ± 1 depending on which of

$$\frac{1-\sum_{i=1}^{k-1} L_{ki}w_i}{L_{kk}} \quad \text{or} \quad \frac{-1-\sum_{i=1}^{k-1} L_{ki}w_i}{L_{kk}}$$

is larger.

After finding w, u is simply computed by

$$\frac{L^{-T}w}{\|L^{-T}w\|_2}.$$

The vector u has the useful property that

$$u^T(M+yI)u \to 0 \quad \text{as} \quad y \to -\lambda_1.$$

5.4 Computing derivatives

The partial derivative of a smooth function $f : \mathbb{R}^n \to \mathbb{R}$ at a given point x can be approximated by the center difference scheme

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x+he_i)-f(x-he_i)}{2h}, \tag{5.1}$$

where e_i is the ith unit vector and $h > 0$ is very small. The gradient can be built using the above formula for $i = 1, 2, \cdots, n$. This process requires evaluation of f at $2n$ points. The optimum value of $he_i \approx \epsilon^{2/3}\ |x_i|e_i$, where ϵ is the machine epsilon [33].

The Hessian of a function can be computed from the following finite difference scheme.

$$\frac{\partial^2 f(x)}{\partial^2 x_i} \approx \frac{f_{i+1,i}-2f_{i,i}+f_{i-1,i}}{h^2}, \tag{5.2}$$

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{f_{i+1,j}+f_{i,j+1}+f_{i-1,j}+f_{i,j-1}-2f_{i,j}-f_{i-1,j+1}-f_{i+1,j-1}}{2h^2}, \tag{5.3}$$

where a typical functional value $f_{i+1,j-1}$ at $(i+1, j-1)$ can be computed by the formula

$$f_{i+1,j-1} = f(x+he_i-he_j).$$

The optimum value of $he_j \approx \sqrt[4]{\epsilon}|x_j|e_j$ [33].

5.5 Numerical error

Since problems will be solved numerically, it is very important to know the effects of underflow, overflow, rounding error and cancellation as described in Chapter 2. Moreover, for control systems, the algorithm must work fast so that output of an algorithm can be used in the real time.

The initial point should be chosen near to the solution for a rapid convergence. The solution of a real problem is often bounded (upper and lower bounds) and hence, the

initial point must lie in the bounds. Often, objective functions, residuals (from nonlinear equations, nonlinear least squares) need scaling for the components of gradient (residual) so that they generate numbers in some desired ranges (say, 1 to 100) and avoid overflow and underflow. Note that scaling is not performed in the implemented algorithms since it depends on the variety of identification problems. The user must use appropriate scales while constructing the objective functions, or residuals.

The initial point x_0 sometimes needs to be scaled. Assume a disproportionate solution (variables differ significantly from each other)

$$x^* \quad = \quad [1.0e-3, 2.0e3, 1.0e5]^T.$$

The initial point x_0 can be scaled by

$$S \quad = \quad diag[1.0e-3, 1.0e3, 1.0e5].$$

However, the solution cannot be known in advance, and therefore scaling factors are determined considering the (possible) bounds of the solution. The scaled initial point z_0 is computed by $z_0 = S^{-1}x_0$. We solve the problem for z instead of x. The solution x^* can be retrieved by

$$x^* := Sz^*.$$

We experience numerical errors, e.g., early termination, high number of iterations, overflow, and underflow from the programs if the variables or gradients (residuals) are disproportionate.

Chapter 6

Test results

6.1 Unconstrained optimization

6.1.1 Test on the behavior of Newton's method

We investigated the convergence behavior of Newton's method on some test functions collected from [17]. We present the test results by specifying the initial points, solution points and required numbers of iterations and by plotting the functions.

1-dimensional functions

In this case, the blue line in the figures represents a function f and the red one indicates its gradient g.

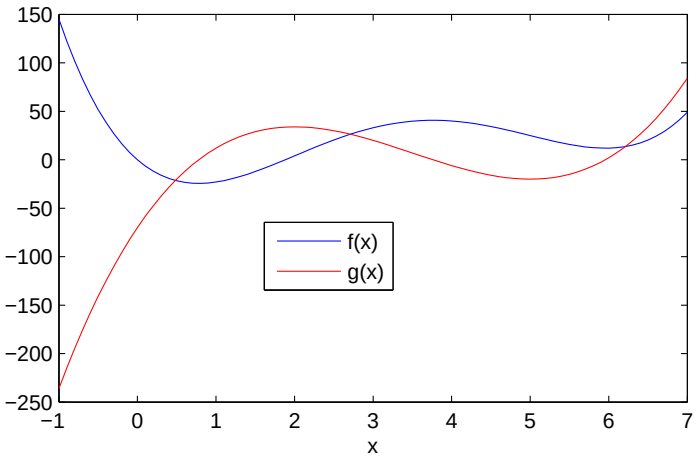

FIGURE 6.1 Plots of a function $f = x^4 - 14x^3 + 60x^2 - 70x$ and its gradient g.

In FIGURE 6.1, a 1-dimensional 4th order function $f = x^4 - 14x^3 + 60x^2 - 70x$ and its gradient g are plotted. The initial point of Newton's method was taken as[-1.0]. The solution we found is [0.78]. It took 3 iterations to converge. For another initial point [5.0], the method converged to [5.957] taking 57 iterations.

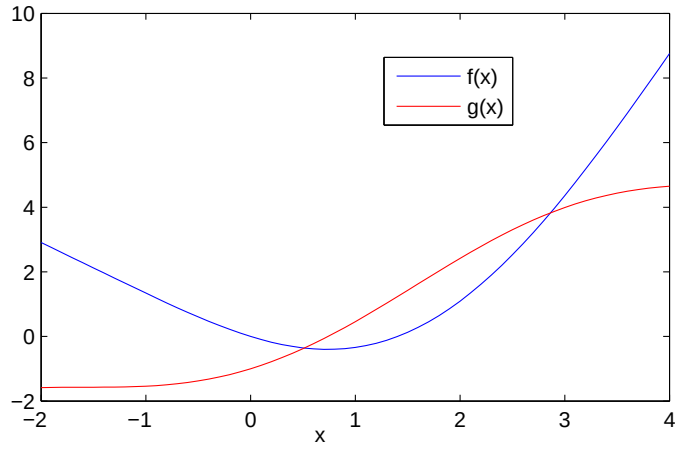

FIGURE 6.2 Plots of a function $f = 0.5x^2 - \sin(x)$ and its gradient g.

FIGURE 6.2 shows a function $f = 0.5x^2 - \sin(x)$ and its gradient g. This function has only one minimum. The solution we obtained by local Newton method is [0.739] where the initial point was [3.0]. It required 5 iterations to converge.

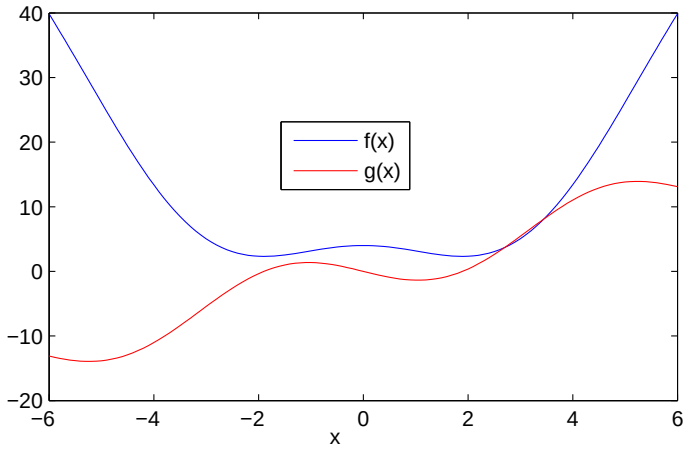

FIGURE 6.3 Plots of a function $f = x^2 + 4\cos(x)$ and its gradient g.

FIGURE 6.3 shows a function $f = x^2 + 4\cos(x)$ and its gradient g. This function has two minima. The initial point was taken as [-2.0]. The solution we obtained is [-1.8955]. It took 4 iterations to converge. The other solution [1.8955] was obtained with the initial point [4.0] and 5 iterations.

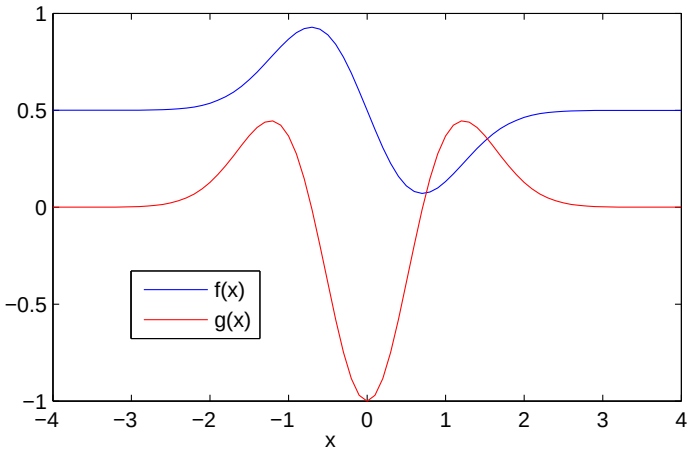

FIGURE 6.4 Plots of a function $f = 0.5 - xe^{-x^2}$ and its gradient g.

FIGURE 6.4 shows the plots of a function $f = 0.5 - xe^{-x^2}$ and its gradient g. For an initial point [0.5], we obtained the solution [0.707] with 3 iterations. But for an initial point [2.0], we obtained the solution [4.26] with 12 iterations where the solution is a maximum. The method failed to converge with an initial point [1.0] because the Hessian became singular after several iterations.

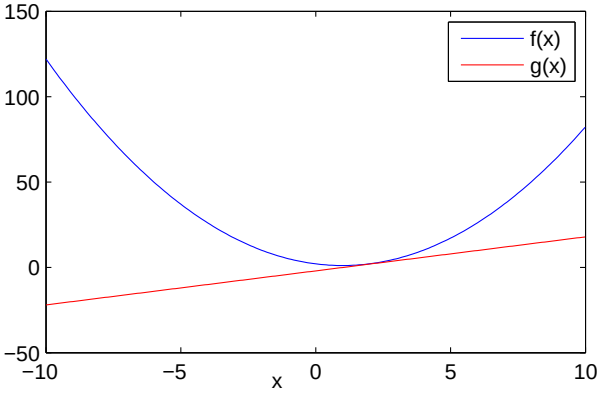

FIGURE 6.5 Plots of a function $f = x^2 - 2x + 2$ and its gradient g.

The initial point was taken as [-2.0] for the quadratic function $f = x^2 - 2x + 2$ shown in FIGURE 6.5. The solution we obtained is [1.0]. It took only 1 iteration to converge.

2-dimensional functions

Now we examine some 2-dimensional functions. Note that in plots of contours, an end-point (enumerated by the number of iterations) of the red dotted line indicates the position of an iterate of Newton's method.

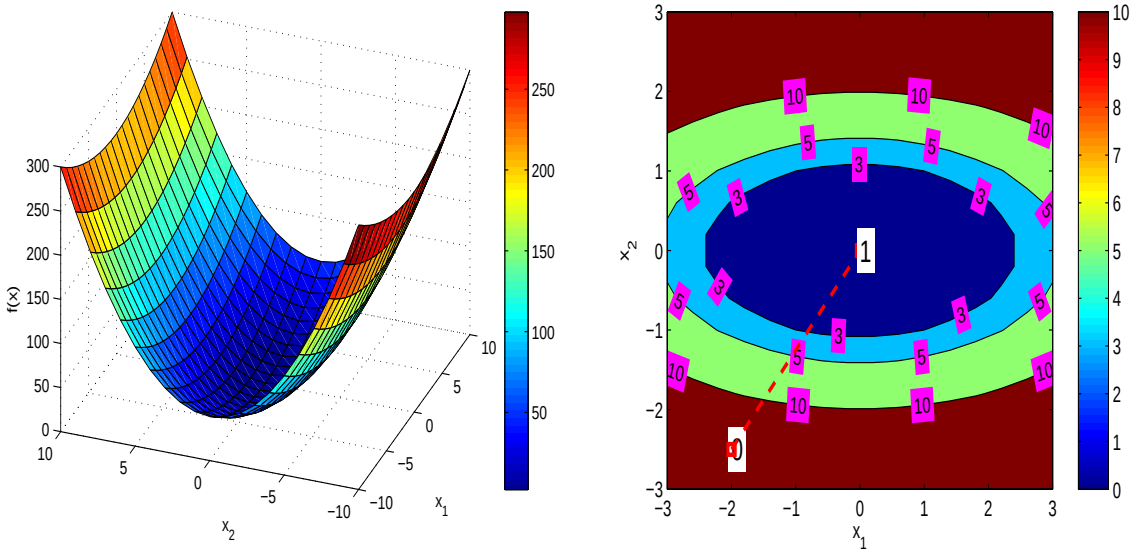

FIGURE 6.6 Plots of the surface and contour of a function $f = 0.5x_1^2 + 2.5x_2^2$.

For the function in FIGURE 6.6 the initial point was taken as [-2, -2.5]. This function is quadratic and therefore has only one minimum. The convergent point is [0,0]. In the right part of the figure we observe that Newton's method took only 1 iteration to converge.

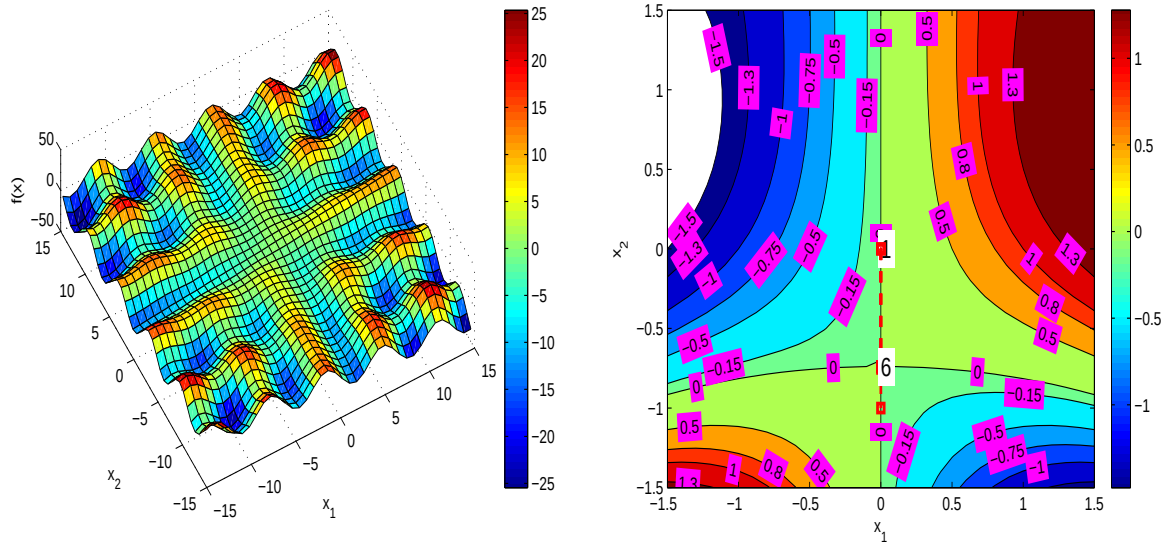

FIGURE 6.7 Plots of the surface and contour of a function $f = x_1 \cos(x_2) + x_2 \sin(x_1)$.

In FIGURE 6.7 a function and its contour are plotted. This function has infinitely many local minima. The initial point lay at [0, 0]. The solution [0, -0.7391] we obtained is a local minimum.

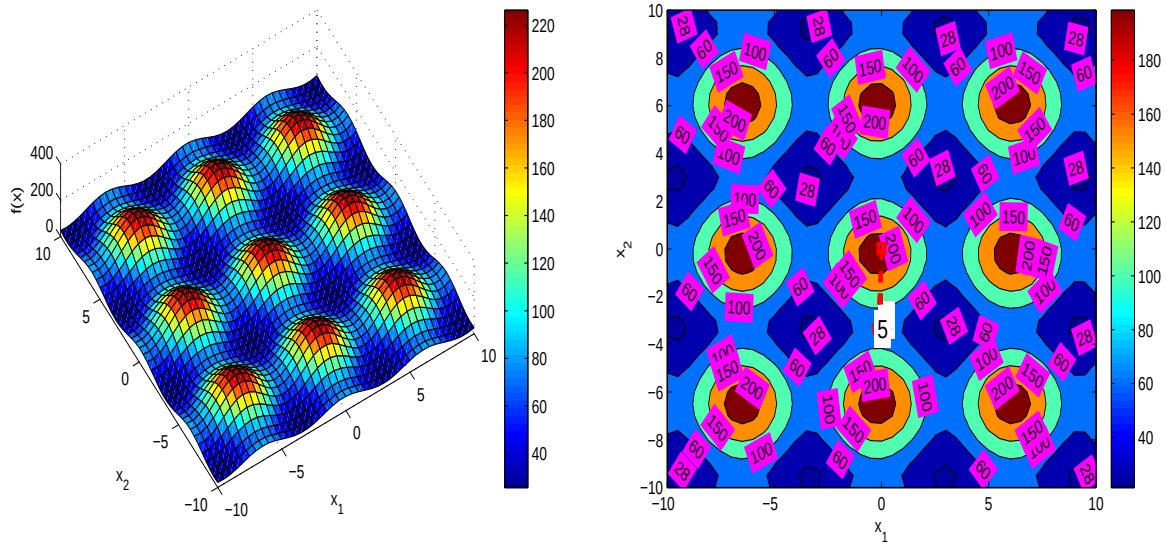

FIGURE 6.8 Plots of the surface and contour of a function $f = (a - a_1 \sin(x_1) + a_2 \cos(x_1))(a - a_1 \sin(x_2) + a_2 \cos(x_2)$ with $a = 10$, $a_1 = 1$ and $a_2 = 5$.

We took the initial point as [0, -3] for the function in FIGURE 6.8. This function has infinitely many local minima. The solution [-0.1974, -3.3390] that we obtained is a saddle point.

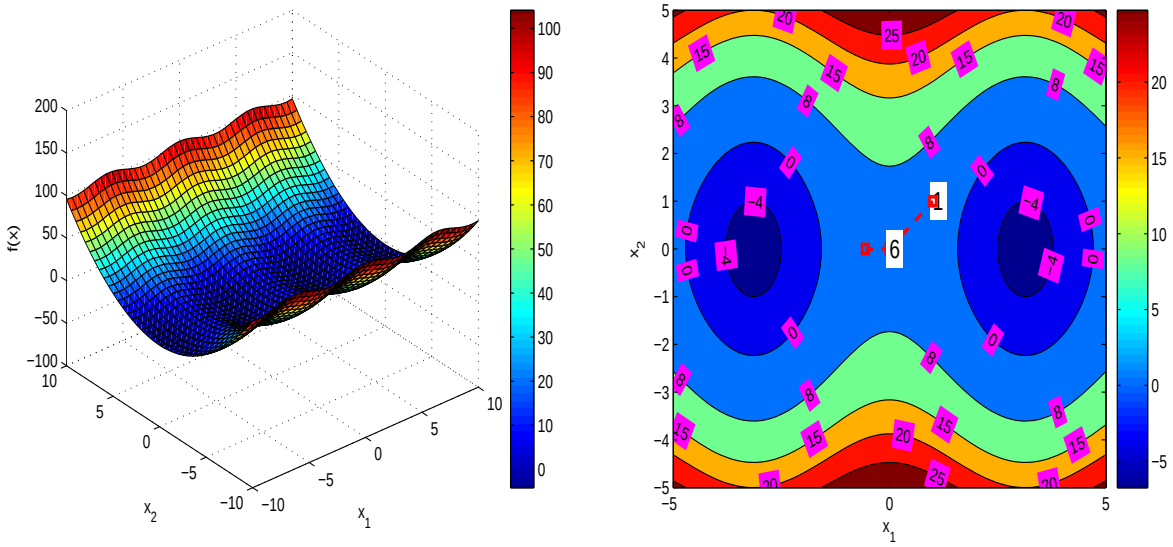

FIGURE 6.9 Plots of the surface and contour of a function $f = 5\cos(x_1) + x_2^2$.

FIGURE 6.9 shows the plots of a function and its contour. This function has infinitely many local minima. The initial point of Newton's method was at [1, 1]. In this case, the obtained solution [0, 0] is a saddle point.

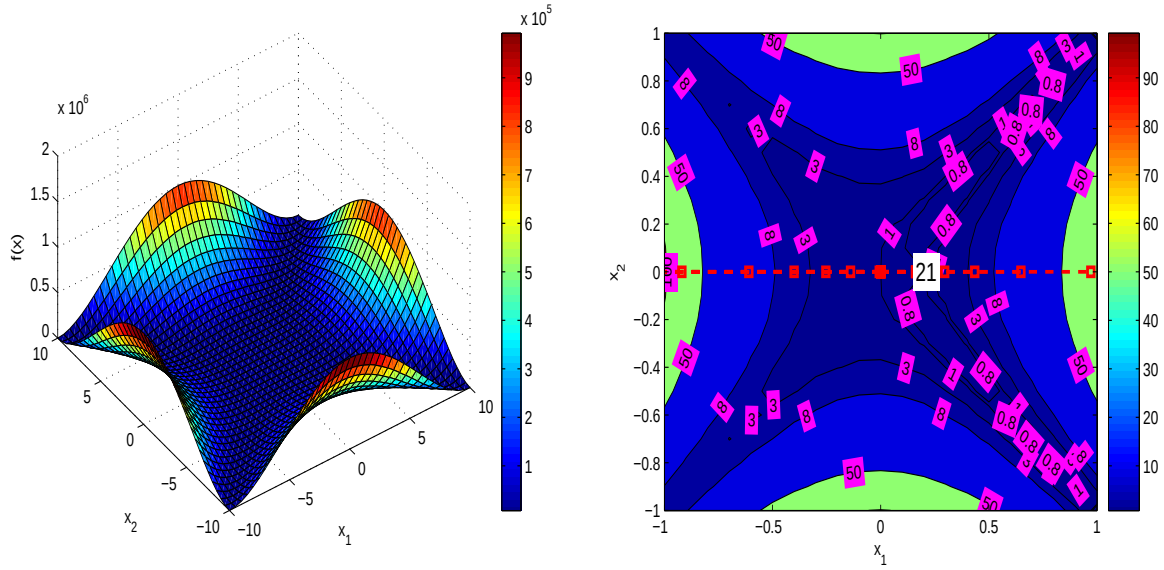

FIGURE 6.10 Plots of the surface and contour of a function $f = 100(x_2^2 - x_1^2)^2 + (1 - x_1)^2$.

FIGURE 6.10 shows a function and its contour. The initial point of Newton's method was taken as [7, 0]. The method converged to the point [0.1613, 0] which is the global minimum.

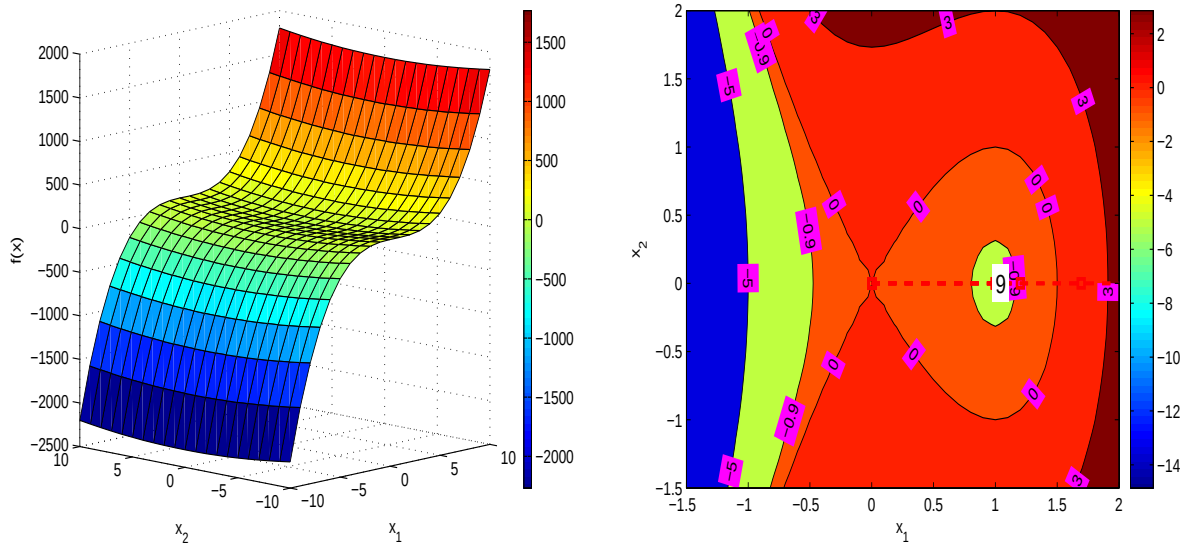

FIGURE 6.11 Plots of the surface and contour of a function $f = 2x_1^3 - 3x_1^2 + x_2^2$.

For the function shown in FIGURE 6.11 the initial point lay at [5, 0]. We obtained the solution [1, 0] which is a local minimum.

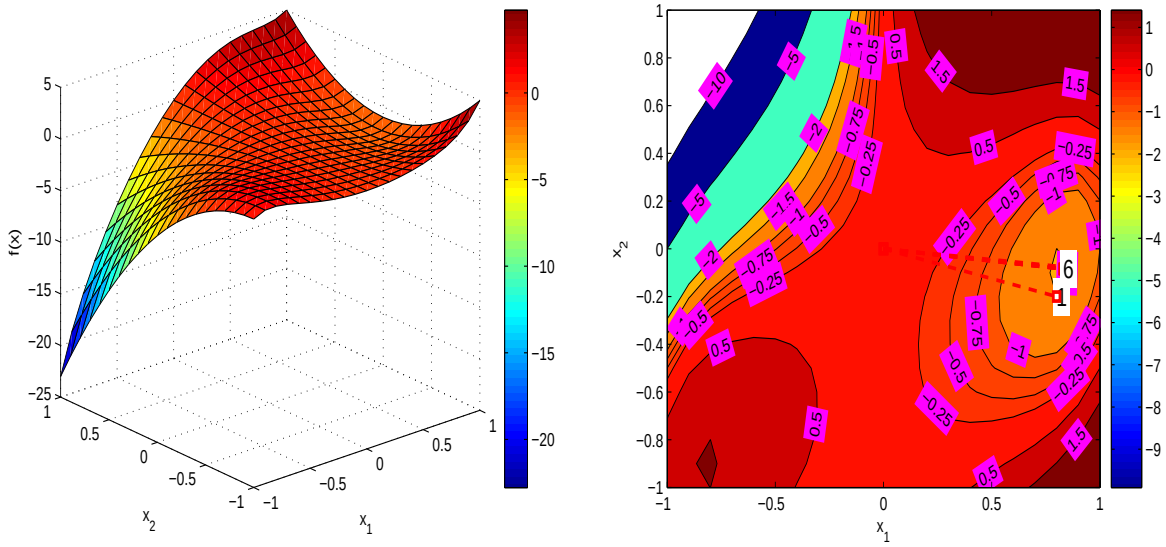

FIGURE 6.12 Plots of the surface and contour of a function $f = 2x_1^7 - 3x_1^2 - 6x_1x_2(x_1 - x_2 - 1)$.

The initial point was taken as [08, -0.2] for the function plotted in FIGURE 6.12. The method converged to the solution [0.8309, -0.0846] which is a local minimum.

6.1.2 Test problems for all variants of Newton's method

We investigated the following test problems [17, 43] by the different variants of Newton's method.

1.
$$f = \sin^2\left(\frac{\pi(x_0+3)}{4}\right) + \left(\frac{\pi(x_{n-1}-1)}{4}\right)^2 + \sum_{i=1}^{n-2}\left[\left(\frac{\pi(x_i-1)}{4}\right)^2\left(1+\sin^2\left(\frac{\pi(x_{i+1}+3)}{4}\right)\right)\right],$$

with $n = 4$.

2.
$$f = \sum_{i=1}^{n-1}\left[1000\left(x_i - x_{i-1}^2\right)^2 + (1-x_{i-1})^2\right],$$

with $n = 4$.

3.
$$f = \sum_{i=1}^{n-1}\left[100\left(x_i - x_{i-1}\right)^2 + (1-x_{i-1})^2\right],$$

with $n = 4$.

4.

$$f = \sum_{i=1}^{n-1} \left[i \left(2x_i - x_{i-1} \right)^2 \right],$$

with $n = 4$.

5.

$$f = \sum_{i=0}^{n} \left\{ \sum_{j=1}^{n-1} \left[\sin(x_i) \sin(x_j) + \sin(x_i) + \sin(x_j) \right]^2 \right\},$$

with $n = 4$.

6.

$$f = \sum_{i=0}^{p-1} \left[x_i - \frac{\sum_{i=0}^{n-1} x_i}{p} - 1 \right]^2 + \sum_{i=p}^{n-1} \left[\frac{\sum_{i=0}^{n-1} x_i}{p} - 1 \right]^2,$$

with $n = 4$, $p = n/2$ and $a = 0.4238537991$.

7.

$$f = \sum_{i=0}^{p-1} (x_{2i} + a)^2 (x_{2i+1} + a)^2 + (x_{2i} + a)^2 + (x_{2i+1} + a)^2,$$

with $n = 4$, $p = n/2$ and $a = 0.4238537991$.

8.

$$\begin{aligned} f &= (3 - 2x_0^2 - 2x_1 + 1)^2 + \sum_{i=1}^{n-1} (3 - 2x_i^2 - x_{i-1} - 2x_{i+1} + 1)^2 \\ &\quad + (3 - 2x_{n-1}^2 - x_{n-2} + 1)^2, \end{aligned}$$

with $n = 4$.

9.

$$\begin{aligned} f &= \left[2x_0 - x_1 + \frac{\left(x_0 + \frac{n+2}{n+1} \right)^3}{2(n+1)^2} \right]^2 + \sum_{i=1}^{n-2} \left[2x_i - x_{i-1} + x_{i+1} + \frac{\left(x_i + \frac{n+i+2}{n+1} \right)^3}{2(n+1)^2} \right]^2 \\ &\quad + \left[2x_{n-1} - x_{n-2} + \frac{\left(x_{n-1} + \frac{2n+1}{n+1} \right)^3}{2(n+1)^2} \right]^2, \end{aligned}$$

with $n = 4$.

The results of test problems with different variants of Newton's method are reported in TABLEs 6.1–6.4. In the tables, the 1st two columns contain the problem number and

the index number (test run count). The 3rd column provides the initial point. In the 3rd column, a single value implies that each component of the initial point holds that value. Columns 4 and 5 give the number of iterations and the functional value. The 6th column contains different criteria (see section 5.1 for detail) satisfied to stop the iterations. We now summarize the stopping criteria in simple terms.

- The Hessian H is singular/indefinite.
- $\|g\|_\infty \approx 0$.
- The ∞-norm of the search direction is approximately 0, i.e., $\|s\|_\infty \approx 0$.
- $\Delta \approx 0$.

Columns 7 and 8 give the convergent point and the *exact solution*, respectively. We call a solution exact when it approximately satisfies the conditions of a minimum (section 3.2).

FIGURE 6.13 shows the performance of each variant of Newton's method in terms of the required number of iterations and the functional value at the solution. We observe that Newton's method with trust-region gives the least functional values while the required iterations are the highest. Newton's method and its line search variant fail when the Hessian matrices become singular or indefinite as a Cholesky decomposition technique was used to solve the linear system arisen in each iteration. The trust-region variant solved all the problems successfully.

TABLE 6.1
Solutions to the test problems of the unconstrained optimization by Newton's method

Example No	Index	Initial point	No. of iterations	f	Criteria	Convergent point	Exact solution
1	1	-10.0	1	30.750	H is indefinite	[-10, -10, -10, -10]	[-11, 1, 1, 1], [5, 1, 1, 1], [9, 1, 1, 1]
	2	0.0	1	0.750	H is indefinite	[0, 0, 0, 0]	
	3	10.0	1	20.750	H is indefinite	[10, 10, 10, 10]	
2	4	-10.0	8	2.27E+006	H is indefinite	[-0.08, 0.277, -3.52, -35.1]	[-1, 1, 1, 1], [1, 1, 1, 1]
	5	0.0	1	2.997	H is indefinite	[0, 0, 0, 0]	
	6	10.0	24	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1, 1]	

TABLE 6.2
Solutions to the test problems of the unconstrained optimization by Newton's method

Example No	Index	Initial point	No. of iterations	f	Criteria	Convergent point	Exact solution
3	7	-10.0	2	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1, 1]	[1, 1, 1, 1, 1]
	8	0.0	1	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1, 1]	
	9	10.0	2	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1, 1]	
4	10	-10.0	2	0.000	$\|g\|_\infty \approx 0$	[-25.26, -12.63, -6.32, -3.16]	[-22.67, -11.33, -5.67, -2.83], [0, 0, 0, 0] [5.44, 2.72, 1.36, 0.68]
	11	0.0	1	0.000	$\|g\|_\infty \approx 0$	[0, 0, 0, 0]	
	12	10.0	2	0.000	H is indefinite	[-5.26, -2.63, -1.32, -0.66]	
5	13	-10.0	1	6.090	H is indefinite	[-10, -10, -10, -10]	[-12.57, -12.57, -12.57, -12.57], [0, 0, 0, 0], [0, 3.14, 15.7, 9.42]
	14	0.0	1	0.000	$\|g\|_\infty \approx 0$	[0, 0, 0, 0, 0]	
	15	10.0	1	23.383	H is indefinite	[10, 10, 10, 10]	
6	16	-10.0	1	164.000	H is singular	[-10, -10, -10, -10]	[-1, -1, -16.8, -16.8], [-1, -1, -7, -7], [-1, -1, 4.12, 4.12]
	17	0.0	1	4.000	H is singular	[0, 0, 0, 0]	
	18	10.0	1	244.000	H is singular	[10, 10, 10, 10]	
7	19	-10.0	1	1.70E+004	H is indefinite	[-10, -10, -10, -10]	[-0.42, -0.42, -0.42, -0.42]
	20	0.0	4	0.000	$\|g\|_\infty \approx 0$	[-0.42, -0.42, -0.42, -0.42]	
	21	10.0	1	2.40E+004	H is indefinite	[10, 10, 10, 10]	
8	22	-10.0	12	0.000	$\|g\|_\infty \approx 0$	[-0.554, -0.639, -0.59, -0.42]	[-0.554, -0.639, -0.59, -0.42], [1.34, 0.58, 0.67, 1.24]
	23	0.0	1	4.000	H is indefinite	[0, 0, 0, 0]	
	24	10.0	12	2.290	$\|g\|_\infty \approx 0$	[1.34, 0.58, 0.67, 1.24]	
9	25	-10.0	11	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.054, -0.035]	[-0.035, -0.052, -0.052, -0.035]
	26	0.0	3	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.054, -0.035]	
	27	10.0	11	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.054, -0.035]	

TABLE 6.3
Solutions to the test problems of the unconstrained optimization by Newton's method with line search

Example No	Index	Initial point	No. of iterations	f	Criteria	Convergent point	Exact solution
1	1	-10.0	1	30.750	H is indefinite	[-10, -10, -10, -10]	[-11, 1, 1, 1], [5, 1, 1, 1], [9, 1, 1, 1]
	2	0.0	1	0.750	H is indefinite	[0, 0, 0, 0]	
	3	10.0	1	20.750	H is indefinite	[10, 10, 10, 10]	
2	4	-10.0	13	2.795	H is indefinite	[-0.31, 0.097, 0.01, 0.001]	[-1, 1, 1, 1], [1, 1, 1, 1]
	5	0.0	2	2.997	H is indefinite	[0, 0, 0, 0]	
	6	10.0	44	0.000	$\Delta \approx 0$	[1, 1, 1, 1, 1]	
3	7	-10.0	3	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1, 1]	[1, 1, 1, 1]
	8	0.0	2	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1, 1]	
	9	10.0	3	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1, 1]	
4	10	-10.0	2	0.000	$\|g\|_\infty \approx 0$	[-25.26, -12.63, -6.32, -3.16]	[-22.67, -11.33, -5.67, -2.83], [0, 0, 0, 0] [5.44, 2.72, 1.36, 0.68]
	11	0.0	1	0.000	$\|g\|_\infty \approx 0$	[0, 0, 0, 0]	
	12	10.0	2	0.000	H is indefinite	[-5.26, -2.63, -1.32, -0.66]	
5	13	-10.0	1	6.090	H is indefinite	[-10, -10, -10, -10]	[-12.57, -12.57, -12.57, -12.57], [0, 0, 0, 0], [0, 3.14, 15.7, 9.42]
	14	0.0	1	0.000	$\|g\|_\infty \approx 0$	[0, 0, 0, 0, 0]	
	15	10.0	1	23.383	H is indefinite	[10, 10, 10, 10]	
6	16	-10.0	1	164.000	H is singular	[-10, -10, -10, -10]	[-1, -1, -16.8, -16.8], [-1, -1, -7, -7], [-1, -1, 4.12, 4.12]
	17	0.0	1	4.000	H is singular	[0, 0, 0, 0]	
	18	10.0	1	244.000	H is singular	[10, 10, 10, 10]	
7	19	-10.0	1	1.70E+004	H is indefinite	[-10, -10, -10, -10]	[-0.42, -0.42, -0.42, -0.42]
	20	0.0	4	0.000	$\|g\|_\infty \approx 0$	[-0.42, -0.42, -0.42, -0.42]	
	21	10.0	1	2.40E+004	H is indefinite	[10, 10, 10, 10]	
8	22	-10.0	12	0.000	$\|g\|_\infty \approx 0$	[-0.554, -0.639, -0.59, -0.42]	[-0.554, -0.639, -0.59, -0.42], [1.34, 0.58, 0.67, 1.24]
	23	0.0	1	4.000	H is indefinite	[0, 0, 0, 0]	
	24	10.0	12	2.290	$\|g\|_\infty \approx 0$	[1.34, 0.58, 0.67, 1.24]	
9	25	-10.0	11	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.054, -0.035]	[-0.035, -0.052, -0.052, -0.035]
	26	0.0	3	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.054, -0.035]	
	27	100.0	11	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.054, -0.035]	

TABLE 6.4
Solutions to the test problems of the unconstrained optimization by Newton's method with trust-region

Example No	Index	Initial point	No. of iterations	f	Criteria	Convergent point	Exact solution
1	1	-10.0	17	0.000	$\|g\|_\infty \approx 0$	[-11, 1, 1, 1]	[-11, 1, 1, 1], [5, 1, 1, 1], [9, 1, 1, 1]
	2	0.0	6	0.000	$\|g\|_\infty \approx 0$	[5, 1, 1, 1]	
	3	10.0	8	0.000	$\|g\|_\infty \approx 0$	[9, 1, 1, 1]	
2	4	-10.0	32	0.000	$\|g\|_\infty \approx 0$	[-1, 1, 1, 1]	[-1, 1, 1, 1], [1, 1, 1, 1]
	5	0.0	31	0.000	$\|g\|_\infty \approx 0$	[-1, 1, 1, 1]	
	6	10.0	37	0.000	$\Delta \approx 0$	[1, 1, 1, 1]	
3	7	-10.0	3	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1]	[1, 1, 1, 1]
	8	0.0	2	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1]	
	9	10.0	3	0.000	$\|g\|_\infty \approx 0$	[1, 1, 1, 1]	
4	10	-10.0	9	0.000	$\|g\|_\infty \approx 0$	[-22.67, -11.33, -5.67, -2.83]	[-22.67, -11.33, -5.67, -2.83], [0, 0, 0, 0], [5.44, 2.72, 1.36, 0.68]
	11	0.0	1	0.000	$\|g\|_\infty \approx 0$	[0, 0, 0, 0]	
	12	10.0	3	0.000	$\|g\|_\infty \approx 0$	[5.44, 2.72, 1.36, 0.68]	
5	13	-10.0	5	0.000	$\|g\|_\infty \approx 0$	[-12.57, -12.57, -12.57, -12.57]	[-12.57, -12.57, -12.57, -12.57], [0, 0, 0, 0], [0, 3.14, 15.7, 9.42]
	14	0.0	1	0.000	$\|g\|_\infty \approx 0$	[0, 0, 0, 0]	
	15	10.0	8	0.000	$\|g\|_\infty \approx 0$	[0, 3.14, 15.7, 9.42]	
6	16	-10.0	3	2.000	$\|g\|_\infty \approx 0$	[-1, -1, -16.8, -16.8]	[-1, -1, -16.8, -16.8], [-1, -1, -7, -7], [-1, -1, 4.12, 4.12]
	17	0.0	2	2.000	$\|g\|_\infty \approx 0$	[-1, -1, -7, -7]	
	18	10.0	3	2.000	$\|g\|_\infty \approx 0$	[-1, -1, 4.12, 4.12]	
7	19	-10.0	11	0.000	$\|g\|_\infty \approx 0$	[-0.42, -0.42, -0.42, -0.42]	[-0.42, -0.42, -0.42, -0.42]
	20	0.0	4	0.000	$\|g\|_\infty \approx 0$	[-0.42, -0.42, -0.42, -0.42]	
	21	10.0	9	0.000	$\|g\|_\infty \approx 0$	[-0.42, -0.42, -0.42, -0.42]	
8	22	-10.0	11	0.000	$\|g\|_\infty \approx 0$	[-0.554, -0.639, -0.59, -0.42]	[-0.554, -0.639, -0.59, -0.42], [1.34, 0.58, 0.67, 1.24]
	23	0.0	5	0.000	$\|g\|_\infty \approx 0$	[-0.554, -0.639, -0.59, -0.42]	
	24	10.0	11	2.290	$\|g\|_\infty \approx 0$	[1.34, 0.58, 0.67, 1.24]	
9	25	-10.0	10	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.052, -0.035]	[-0.035, -0.052, -0.052, -0.035]
	26	0.0	3	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.052, -0.035]	
	27	10.0	11	0.000	$\|g\|_\infty \approx 0$	[-0.035, -0.052, -0.052, -0.035]	

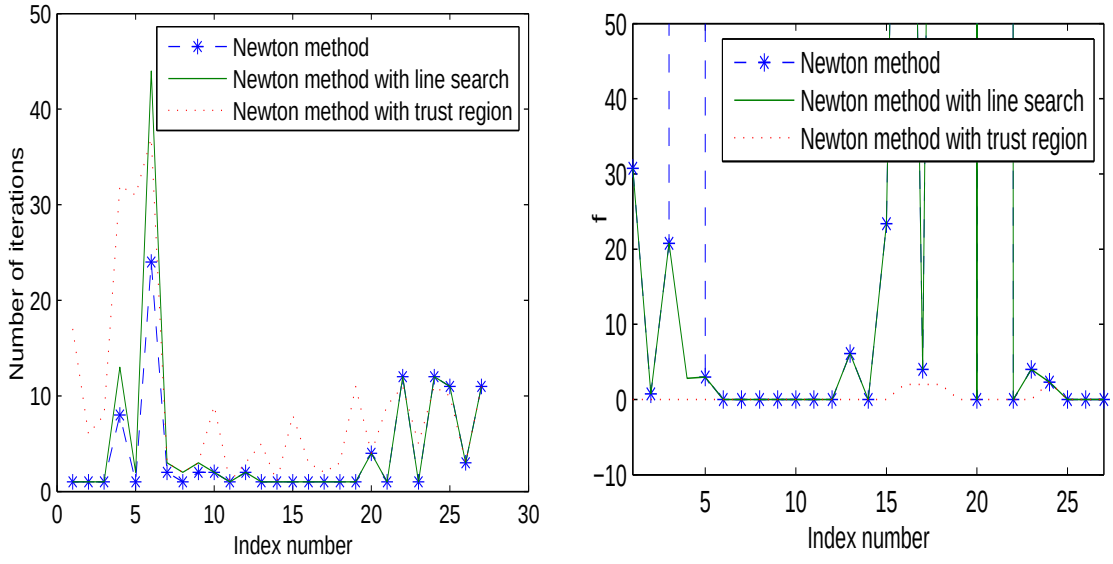

FIGURE 6.13 Number of iterations and norm of residual versus test run count

6.2 Nonlinear system of equations

We investigated the following test problems of nonlinear system of equations [17] by the different variants of Newton's method.

1.

$$\begin{aligned} x_0^2 - x_1 &= -0.25 \\ x_1^2 - x_0 &= -0.25 \end{aligned}$$

2.

$$\begin{aligned} x_0 - 2x_1 &= 2 \\ x_0^2 + 4x_1^2 &= 4 \end{aligned}$$

3.

$$\begin{aligned} 2\sin(x_0) + \cos(x_1) - 5x_0 &= 0 \\ 4\cos(x_0) + \sin(x_1) - 5x_1 &= 0 \end{aligned}$$

4.

$$\begin{aligned} x_0 + x_1(x_1(5 - x_1) - 2) - 13 &= 0 \\ x_0 + x_1(x_1(1 + x_1) - 14) - 29 &= 0 \end{aligned}$$

5.

$$\begin{aligned} x_0^2 + x_1^2 + x_2^2 - 5 &= 0 \\ x_0 + x_1 - 1 &= 0 \\ x_0 + x_2 - 3 &= 0 \end{aligned}$$

6.

$$\begin{aligned} x_0 + 10x_1 - 5 &= 0 \\ x_2 + x_3 - 2 &= 0 \\ (x_1 - x_2)^2 - 10 &= 0 \\ (x_0 - x_3)^2 - 10 &= 0 \end{aligned}$$

7.

$$\begin{aligned} x_0 &= 0 \\ \frac{10x_0}{x_0 + 0.1} + 2x_1^2 &= 0 \end{aligned}$$

8.

$$\begin{aligned} 1000x_0x_1 - 1 &= 0 \\ e^{-x_0} + e^{-x_1} - 1.0001 &= 0 \end{aligned}$$

TABLEs 6.5–6.7 show the results obtained from above test problems by different variants of Newton's method. In the tables, the 1st two columns contain the problem number and the index number (test run count). The 3rd column provides the initial point. In the 3rd column, a single value implies that each component of the initial point holds that value. Columns 4 and 5 give the number of iterations and the norm of the residual. The 6th column contains different criteria (see section 5.2 for detail) satisfied to stop the iterations. We now summarize the stopping criteria in the following simple terms.

- The Jacobian A is singular.
- The ∞-norm of the residual is approximately 0, i.e., $\|F(x)\|_\infty \approx 0$.
- The ∞-norm of the search direction is approximately 0, i.e., $\|s\|_\infty \approx 0$.
- $\Delta \approx 0$.

Columns 7 and 8 give the convergent point and the *exact solution*, respectively. We call a solution x exact when it satisfies $F(x) \approx 0$. The left part of FIGURE 6.14 shows the required number of iterations while its right part give the ∞-norm of the residual for each variant of Newton's method. We check that Newton's method with trust-region gives the

least norm solutions while the required iterations are the highest. Newton's method and its line search variant fail when the Jacobian matrices become singular. The trust-region variant solved most problems successfully.

TABLE 6.5
Solutions to the test problems of the nonlinear system of equations by Newton's method

Example No	Index	Initial point	No. of iterations	Norm of residual	Stopping criteria	Convergent point	Exact solution
1	1	-10.0	15	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	[0.5, 0.5]
	2	0.0	10	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	
	3	10.0	15	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	
2	4	-10.0	10	0.000	$\|F\|_\infty \approx 0$	[2, 0]	[2, 0], [0, 1]
	5	0.0	33	0.000	$\|F\|_\infty \approx 0$	[0, 1]	
	6	10.0	10	0.000	$\|F\|_\infty \approx 0$	[0, 1]	
3	7	-10.0	7	0.000	$\|F\|_\infty \approx 0$	[0.1937, 0.947]	[0.1937, 0.947]
	8	0.0	5	0.000	$\|F\|_\infty \approx 0$	[0.1937, 0.947]	
	9	10.0	6	0.000	$\|s\|_\infty \approx 0$	[0.1937, 0.947]	
4	10	-10.0	14	0.000	$\|F\|_\infty \approx 0$	[5, 4]	[5, 4]
	11	0.0	23	0.000	$\|F\|_\infty \approx 0$	[5, 4]	
	12	10.0	8	0.000	$\|F\|_\infty \approx 0$	[5, 4]	
	13	[15, -2]	55	0.000	$\|F\|_\infty \approx 0$	[5, 4]	
5	14	-10.0	11	0.000	$\|F\|_\infty \approx 0$	[1.666, -0.666, 1.333]	[1.666, -0.666, 1.333], [1.666, -0.666, 1.333], [1, 0, 2]
	15	0.0	1	5.916	A is singular	[0, 0, 0]	
	16	10.0	10	0.000	$\|s\|_\infty \approx 0$	[1, 0, 2]	
6	17	-10.0	30	0.000	$\|F\|_\infty \approx 0$	[-1.363, 0.636, 3.798, 1.798]	[-1.363, 0.636, -2.526, -4.526], [-1.363, 0.636, 3.798, 1.798]
	18	0.0	1	15.130	A is singular	[0, 0, 0, 0]	
	19	10.0	30	0.000	$\|F\|_\infty \approx 0$	[-1.363, 0.636, 3.798, 1.798]	
7	20	-10.0	15	0.000	$\|F\|_\infty \approx 0$	[0, 0]	[0, 0]
	21	0.0	1	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
	22	10.0	15	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
	23	[1.8, 0]	39	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
8	24	-10.0	2	-	A is singular	Not convergent	[9.106, 1.098e-5], [1.09e-5, 9.106]
	25	0.0	1	1.414	A is singular	[0, 0, 0]	
	26	10.0	2	-	A is singular	Not convergent	
	27	[0, 1]	12	0.000	$\|F\|_\infty \approx 0$	[1.09e-5, 9.106]	

TABLE 6.6
Solutions to the test problems of the nonlinear system of equations by Newton's method with line search

Example No	Index	Initial point	No. of iterations	Norm of residual	Stopping criteria	Convergent point	Exact solution
1	1	-10.0	15	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	[0.5, 0.5]
	2	0.0	10	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	
	3	10.0	15	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	
2	4	-10.0	10	0.000	$\|F\|_\infty \approx 0$	[2, 0]	[2, 0], [0, 1]
	5	0.0	7	4.472	$\Delta \approx 0$	[0, 0]	
	6	10.0	9	0.000	$\|s\|_\infty \approx 0$	[0, 1]	
3	7	-10.0	7	0.000	$\|F\|_\infty \approx 0$	[0.1937, 0.947]	[0.1937, 0.947]
	8	0.0	5	0.000	$\|F\|_\infty \approx 0$	[0.1937, 0.947]	
	9	10.0	6	0.000	$\|s\|_\infty \approx 0$	[0.1937, 0.947]	
4	10	-10.0	16	7.706	$\Delta \approx 0$	[13.14, -0.936]	[5, 4]
	11	0.0	14	8.925	$\Delta \approx 0$	[16.994, -0.76]	
	12	10.0	8	0.000	$\|F\|_\infty \approx 0$	[5, 4]	
	13	[15, -2]	13	7.710	$\Delta \approx 0$	[11.805, -1.022]	
5	14	-10.0	10	0.000	$\|F\|_\infty \approx 0$	[1.666, -0.666, 1.333]	[1.666, -0.666, 1.333], [1.666, -0.666, 1.333], [1, 0, 2]
	15	0.0	1	5.916	A is singular	[0, 0, 0]	
	16	10.0	10	0.000	$\|F\|_\infty \approx 0$	[1, 0, 2]	
6	17	10.0	7	0.000	$\|F\|_\infty \approx 0$	[-1.363, 0.636, 3.798, 1.798]	[-1.363, 0.636, -2.526, -4.526], [-1.363, 0.636, 3.798, 1.798]
	18	0.0	30	15.130	A is singular	[0, 0, 0, 0]	
	19	10.0	7	0.000	$\|F\|_\infty \approx 0$	[-1.363, 0.636, 3.798, 1.798]	
7	20	-10.0	15	0.000	$\|F\|_\infty \approx 0$	[0, 0]	[0, 0]
	21	0.0	1	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
	22	10.0	15	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
	23	[1.8, 0]	6	9.640	$\Delta \approx 0$	[1.8, 0]	
8	24	-10.0	23	1.330	$\Delta \approx 0$	[-0.077, -0.00028]	[9.106, 1.098e-5], [1.09e-5, 9.106]
	25	0.0	1	1.414	A is singular	[0, 0, 0]	
	26	10.0	2	-	A is singular	Not convergent	
	27	[0, 1]	12	0.000	$\|F\|_\infty \approx 0$	[1.09e-5, 9.106]	

TABLE 6.7
Solutions to the test problems of the nonlinear system of equations by Newton's method with trust-region

Example No	Index	Initial point	No. of iterations	Norm of residual	Stopping criteria	Convergent point	Exact solution
1	1	-10.0	11	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	[0.5, 0.5]
	2	0.0	6	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	
	3	10.0	11	0.000	$\|F\|_\infty \approx 0$	[0.5, 0.5]	
2	4	-10.0	9	0.000	$\|s\|_\infty \approx 0$	[2, 0]	[2, 0], [0, 1]
	5	0.0	6	0.000	$\|s\|_\infty \approx 0$	[0, 1]	
	6	10.0	8	0.000	$\|F\|_\infty \approx 0$	[0, 1]	
3	7	-10.0	6	0.000	$\|F\|_\infty \approx 0$	[0.1937, 0.947]	[0.1937, 0.947]
	8	0.0	4	0.000	$\|F\|_\infty \approx 0$	[0.1937, 0.947]	
	9	10.0	7	0.000	$\|s\|_\infty \approx 0$	[0.1937, 0.947]	
4	10	-10.0	21	6.990	$\Delta \approx 0$	[11.41, -0.896]	[5, 4]
	11	0.0	27	6.990	$\Delta \approx 0$	[11.41, -0.896]	
	12	10.0	7	0.000	$\|F\|_\infty \approx 0$	[5, 4]	
	13	[15, -2]	34	6.990	$\Delta \approx 0$	[11.41, -0.896]	
5	14	-10.0	9	0.000	$\|F\|_\infty \approx 0$	[1.666, -0.666, 1.333]	[1.666, -0.666, 1.333], [1.666, -0.666, 1.333], [1, 0, 2]
	15	0.0	7	0.000	$\|F\|_\infty \approx 0$	[1.666, -0.666, 1.333]	
	16	10.0	9	0.000	$\|s\|_\infty \approx 0$	[1, 0, 2]	
6	17	-10.0	6	0.000	$\|F\|_\infty \approx 0$	[-1.363, 0.636, -2.526, -4.526]	[-1.363, 0.636, -2.526, -4.526], [-1.363, 0.636, 3.798, 1.798]
	18	0.0	5	0.000	$\|F\|_\infty \approx 0$	[-1.363, 0.636, -2.526, -4.526]	
	19	10.0	7	0.000	$\|s\|_\infty \approx 0$	[-1.363, 0.636, 3.798, 1.798]	
7	20	-10.0	17	0.000	$\|F\|_\infty \approx 0$	[0, 0]	[0, 0]
	21	0.0	1	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
	22	10.0	17	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
	23	[1.8, 0]	18	0.000	$\|F\|_\infty \approx 0$	[0, 0]	
8	24	-10.0	33	1.020	$\|s\|_\infty \approx 0$	[-0.01, -0.01]	[9.106, 1.098e-5], [1.09e-5, 9.106]
	25	0.0	47	0.000	$\|s\|_\infty \approx 0$	[9.106, 1.098e-5]	
	26	10.0	6	0.000	$\|F\|_\infty \approx 0$	[9.106, 1.098e-5]	
	27	[0, 1]	28	0.000	$\|F\|_\infty \approx 0$	[1.09e-5, 9.106]	

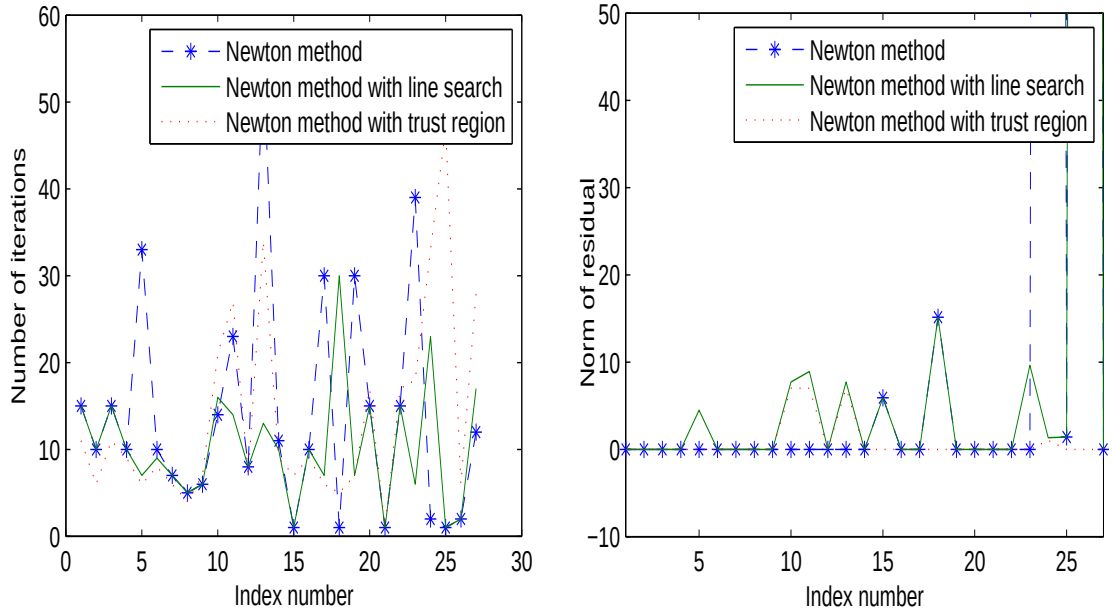

FIGURE 6.14 Number of iterations and norm of residual versus test run count

6.3 Nonlinear least squares

We investigated the following test problems [17] by the different variants of Gauss-Newton method.

1.

$$z(t, x) = x_0 \sin(t + x_1),$$

with $y = 1, 3, 9$ corresponding to $t = 5, 4, 6$.

2.

$$z(t, x) = t \sin(x_0) \cos(t + x_1) + t^2 x_2,$$

with $y = 10, 4, 2, 1$ corresponding to $t = 1, 4, 9, 6$.

3.

$$z(t, x) = x_0^3 t^2 + x_0 x_1 \sin(t) + x_1 x_2 t^3,$$

with $y = 10, 4, 2, 1$ corresponding to $t = 1, 4, 9, 6$.

4.

$$z(t, x) = x_0 e^{x_1 t},$$

with $y = 2, 0.7, 0.3, 0.1$ corresponding to $t = 0, 1, 2, 3$.

5.

$$z(t, x) = x_0 + x_1 t + x_2 t^2 + x_3 e^{x_4 t},$$

with $y = 20, 51.58, 68.73, 75.46, 74.36, 67.09, 54.73, 37.98, 17.28$ corresponding to $t =$ 0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2.

6.

$$z(t, x) = \frac{x_0}{1 + x_1/t},$$

with $y = 0.024, 0.036, 0.053, 0.06, 0.064$ corresponding to $t =$2.5, 5, 10, 15, 20.

7.

$$z(t, x) = t^2 x_0 + t(\sin(x_1) + x_2^3),$$

with $y = 16.5, 150.6, 263.1, 24.7, 208.5, 9.9, 2.7, 163.9, 325.0, 54.3$ corresponding to t = 3.6, 7.7, 9.3, 4.1, 8.6, 2.8, 1.3, 7.9, 10, 5.4.

We present the results obtained from above test problems by different variants of Newton's method in TABLEs 6.8–6.10. The 1st two columns of the tables contain the problem number and the index number (test run count). The 3rd column provides the initial point. In the 3rd column, a single value implies that each component of the initial point holds that value. Columns 4 and 5 give the number of iterations and the norm of the residual. The 6th column contains different criteria (see section 5.2 for detail) satisfied to stop the iterations. We now summarize the stopping criteria in the following simple terms.

- The Jacobian A is singular.
- The ∞-norm $\|A^T F\|_\infty \approx 0$ where F is the rasidual.
- The ∞-norm $\|s\|_\infty \approx 0$.
- $\Delta \approx 0$.

Columns 7 and 8 give the convergent point and the *exact solution*, respectively. We call a solution x exact when it satisfies $A^T F \approx 0$. FIGURE 6.15 plots the required number of iterations and the ∞-norm of the residual against the test run count for each variant of Newton's method. We note that Newton's method with trust-region gives the least norm solutions while the required iterations are moderate. Newton's method and its line search variant fail when the Jacobian matrices become singular. The trust-region variant solved almost all problems successfully.

TABLE 6.8
Solutions to the test problems of the nonlinear least squares by Newton's method

Example No	Index	Initial point	No. of iterations	Norm of residual	Criteria	Convergent point	Exact solution
1	1	-10.0	6	6.130	$\|s\|_\infty \approx 0$	[5.92, -10.359]	[-5.92, -0.93]
	2	0.0	7	6.130	$\|s\|_\infty \approx 0$	[-5.92, -0.93]	
	3	10.0	8	6.130	$\|s\|_\infty \approx 0$	[5.92, -4.075]	
2	4	-10.0	100	17.832	Maximum iteration reached	[-8.644, -9.434, -0.209]	[1.57, 0, -0.09]
	5	0.0	100	18.990	Maximum iteration reached	[2.2, 0, -0.209]	
	6	10.0	100	31.025	Maximum iteration reached	[10.96, 8.4, -0.209]	
3	7	-10.0	28	6.573	$\|s\|_\infty \approx 0$	[0.855, 9.406, -0.0076]	[0.855, 9.4, -0.007]
	8	0.0	1	11.000	$\|s\|_\infty \approx 0$	[0, 0, 0]	
	9	10.0	11	6.573	$\|s\|_\infty \approx 0$	[0.855, 9.406, -0.0076]	
4	10	-10.0	2	0.768	$\|s\|_\infty \approx 0$	[2, -1551.71]	[1.995, -1.009]
	11	0.0	7	0.044	$\|s\|_\infty \approx 0$	[1.995, -1.009]	
	12	10.0	9	0.768	$\|s\|_\infty \approx 0$	[2, -749.95]	
5	13	-10.0	100	-	Maximum iteration reached	Not convergent	[109.73, 5.23, -25.054, -89.73, -1.75]
	14	-5.0	15	5.044	$\|s\|_\infty \approx 0$	[35.27, 87.03, -48.48, -15.27, -1684.38]	
	15	0.0	3	10.220	$\|s\|_\infty \approx 0$	[12.59, 105.90, -55.89, 12.59, 0.0]	
	16	5.0	100	-	Maximum iteration reached	Not convergent	
	17	10.0	9	5.044	$\|s\|_\infty \approx 0$	[35.27, 87.03, -48.48, -15.27, -92.48]	
6	18	-10.0	100	-	Maximum iteration reached	Not convergent	[0.086, 6.562]
	19	0.0	8	0.002	$\|s\|_\infty \approx 0$	[0.086, 6.562]	
	20	10.0	8	0.002	$\|s\|_\infty \approx 0$	[0.086, 6.562]	
7	21	-10.0	100	-	Maximum iteration reached	[2.93, 4.8e+6, -1213]	[4.366, 4.088, -2.274]
	22	0.0	30	268.18	A is singular	[4.37, -377, 0]	
	23	10.0	19	27.83	$\|s\|_\infty \approx 0$	[4.37, -3.2e+6, -2.32]	

TABLE 6.9
Solutions to the test problems of the nonlinear least squares by Newton's method with line search

Example No	Index	Initial point	No. of iterations	Norm of residual	Criteria	Convergent point	Exact solution
1	1	-10.0	6	6.130	$\|s\|_\infty \approx 0$	5.92, -10.359	[-5.92, -0.93]
	2	0.0	7	6.130	$\|s\|_\infty \approx 0$	-5.92, -0.93	
	3	10.0	8	6.130	$\|s\|_\infty \approx 0$	5.92, -4.075	
2	4	-10.0	14	14.180	$\Delta \approx 0$	-7.85, -9.42, -0.209	[1.57, 0, -0.09]
	5	0.0	12	9.870	$\Delta \approx 0$	1.31, 0, -0.12	
	6	10.0	12	14.190	$\Delta \approx 0$	10.97, 9.41, -0.209	
3	7	-10.0	18	18.070	$\|s\|_\infty \approx 0$	-0.6, -10.37, -0.001	[0.855, 9.4, -0.007]
	8	0.0	1	11.000	$\|s\|_\infty \approx 0$	0, 0, 0	
	9	10.0	11	6.573	$\|s\|_\infty \approx 0$	0.855, 9.406, -0.0076	
4	10	-10.0	2	0.768	$\|s\|_\infty \approx 0$	2, -1551.71	[1.995, -1.009]
	11	0.0	7	0.044	$\|s\|_\infty \approx 0$	1.995, -1.009	
	12	10.0	16	2.141	$\Delta \approx 0$	0, 3.98	
5	13	-10.0	12	263.130	$\Delta \approx 0$	4.256, 9.08, -17.8, -14.25,-0.358	[109.73, 5.23, -25.054, -89.73, -1.75]
	14	-5.0	12	208.904	$\Delta \approx 0$	12.68, 8.54, -12.83, -14.86, 0.194	
	15	0.0	3	10.220	$\|s\|_\infty \approx 0$	12.59, 105.90, -55.89, 12.59, 0.0	
	16	5.0	14	13.970	$\Delta \approx 0$	21.11, 134.68, -83.33, 0.059, 3.46	
	17	10.0	15	32.040	$\Delta \approx 0$	22, 126.2, -71.87, 0, 5.78	
6	18	-10.0	100	-	Maximum iteration reached	Not convergent	[0.086, 6.562]
	19	0.0	8	0.002	$\|s\|_\infty \approx 0$	0.086, 6.562	
	20	10.0	8	0.002	$\|s\|_\infty \approx 0$	0.086, 6.562	
7	21	-10.0	11	23197	$\|s\|_\infty \approx 0$	-10, 6.39, -9.96	[4.366, 4.088, -2.274]
	22	0.0	11	247.33	$\Delta \approx 0$	4.36, -14.14, 0	
	23	10.0	9	27.83	$\|s\|_\infty \approx 0$	4.37, -7134, -2.34	

TABLE 6.10
Solutions to the test problems of the nonlinear least squares by Newton's method with trust-region

Example No	Index	Initial point	No. of iterations	Norm of residual	Criteria	Convergent point	Exact solution
1	1	-10.0	8	6.130	$\|A^T F\|_\infty \approx 0$	[5.92, -10.359]	[-5.92, -0.93]
	2	0.0	9	6.130	$\|s\|_\infty \approx 0$	[-5.92, -0.93]	
	3	10.0	8	6.130	$\|A^T F\|_\infty \approx 0$	[5.92, 8.49]	
2	4	-10.0	21	9.370	$\|s\|_\infty \approx 0$	[1.57, -12.56, -0.09]	[1.57, 0, -0.09]
	5	0.0	23	9.370	$\Delta \approx 0$	[1.57, 0, -0.09]	
	6	10.0	23	9.370	$\Delta \approx 0$	[10.99, 9.42, -0.09]	
3	7	-10.0	100	9.760	$\Delta \approx 0$	[-0.008, -440.08, 0]	[0.855, 9.4, -0.007]
	8	0.0	0	11.000	$\|A^T F\|_\infty \approx 0$	[0, 0, 0]	
	9	10.0	11	6.573	$\|s\|_\infty \approx 0$	[0.855, 9.4, -0.007]	
4	10	-10.0	8	0.044	$\|A^T F\|_\infty \approx 0$	[1.995, -1.009]	[1.995, -1.009]
	11	0.0	7	0.044	$\|A^T F\|_\infty \approx 0$	[1.995, -1.009]	
	12	10.0	20	0.044	$\|A^T F\|_\infty \approx 0$	[1.995, -1.009]	
5	13	-10.0	12	0.003	$\|A^T F\|_\infty \approx 0$	[109.73, 5.23, -25.054, -89.73, -1.75]	[109.73, 5.23, -25.054, -89.73, -1.75]
	14	-5.0	15	0.003	$\|A^T F\|_\infty \approx 0$	[109.73, 5.23, -25.054, -89.73, -1.75]	
	15	0.0	100	2.923	Maximum iteration reached	[-200.84, 1.8, -136.6, 221.93, 0.61]	
	16	5.0	58	0.003	$\|A^T F\|_\infty \approx 0$	[109.73, 5.23, -25.054, -89.73, -1.75]	
	17	10.0	43	0.003	$\|s\|_\infty \approx 0$	[109.73, 5.23, -25.054, -89.73, -1.75]	
6	18	-10.0	100	-	Maximum iteration reached	Not convergent	[0.086, 6.562]
	19	0.0	6	0.002	$\|A^T F\|_\infty \approx 0$	[0.086, 6.562]	
	20	10.0	5	0.002	$\|A^T F\|_\infty \approx 0$	[0.086, 6.562]	
7	21	-10.0	30	27.83	$\|s\|_\infty \approx 0$	[4.36, -19.69, -2.27]	[4.366, 4.088, -2.274]
	22	0.0	20	61.12	$\Delta \approx 0$	[3.02, -7.85, 0]	
	23	10.0	30	27.83	$\|s\|_\infty \approx 0$	[4.366, 4.088, -2.274]	

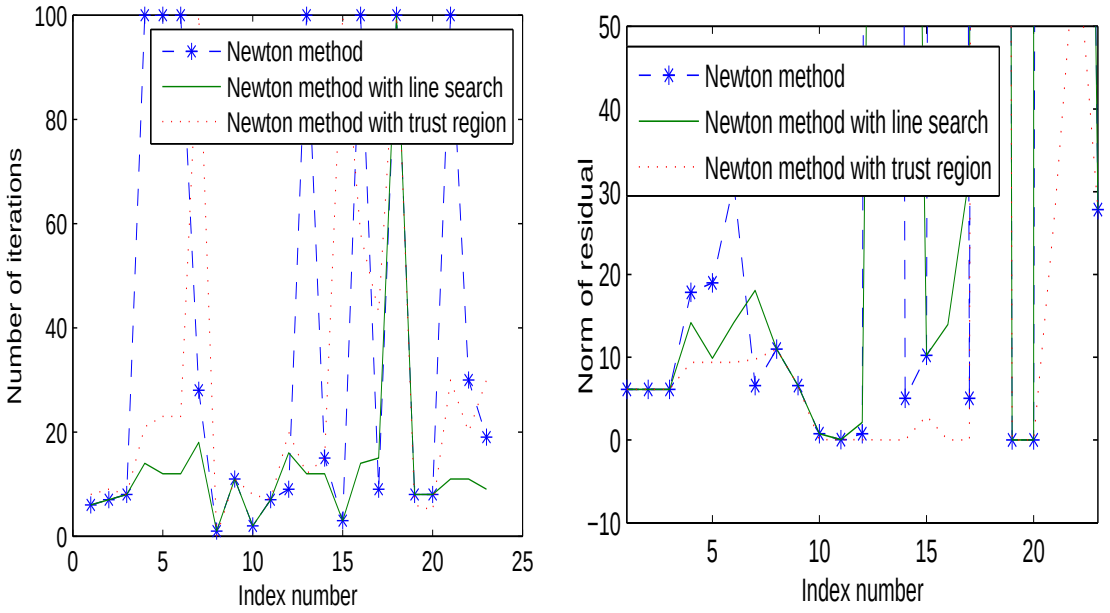

FIGURE 6.15 Number of iterations and norm of residual versus test run count

Chapter 7

Identification of combustion parameters in a diesel engine

7.1 Introduction

In a diesel engine, three combustion parameters which describe the crank angle at the start of ignition, the total amount of heat release and the combustion duration can be used to control the speed, efficiency, noise and exhaust emission. Pressure analysis of a diesel engine cylinder can be used for identifying these parameters. Advanced measurement facilities are now available for the precise acquisition of the cylinder pressure signal. The combustion parameters can be determined using this pressure signal and the pressure derived from a mathematical model of the combustion process. One such mathematical model of the combustion process called the *Wiebe function* [38] is widely used in application. I investigated the Wiebe function and the analysis shows that the computational time to determine the combustion parameters is too high to be used in the real time operation. That is why I propose several alternative models which our investigation proves to be cheap in computation. Thus these models provide suitable choices for the real time operation.

7.2 Diesel engine

FIGURE 7.1 shows geometric parameters of a diesel engine where "TDC" is the *top dead center* and "BDC" is the *bottom dead center*. The top dead center refers to the position of the piston at the crank angle 0^o where cylinder volume V_0 is the minimum. The position of the piston at the crank angle 180^o is the bottom dead center. At this position the cylinder volume is the maximum. We now give a simple description of a four-stroke Diesel engine.

In the first stroke, the piston moves from top to bottom (BDC) and air is drawn through the intake valve in the cylinder. The next stroke begins when the piston starts its upwards movement and the intake valve closes. As the piston moves up, air is compressed and heated tremendously. When the piston is near TDC, fuel is injected into the cylinder.

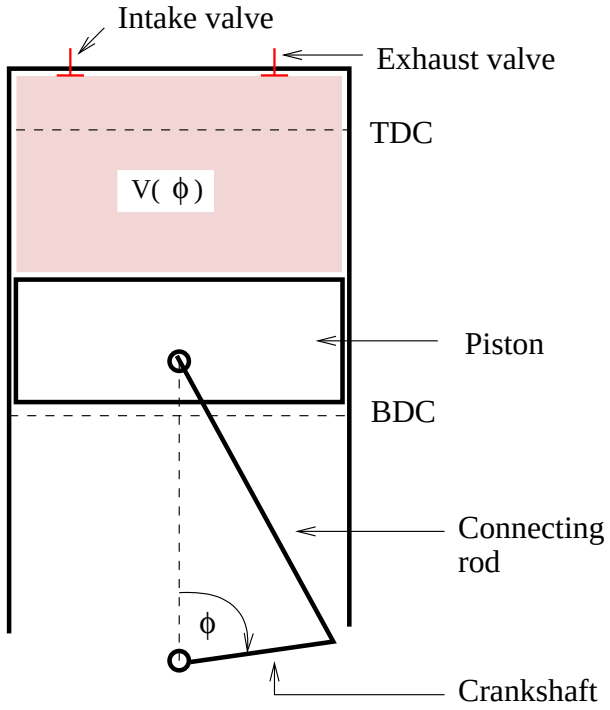

FIGURE 7.1 Geometric parameters of a diesel engine

High temperature of the compressed air in the cylinder ignites the fuel vapor resulting in a small explosion. In the next stroke, this explosion forces the piston back downwards to BDC.

In the last stroke, the exhaust valve opens, and the cylinder is swept clean of burnt fuel by the piston moving in the cylinder to TDC. A connecting rod transmits this motion to a crankshaft to convert linear motion to rotary motion for use as power in a variety of applications. This entire cycle is repeated for every two revolutions of the crankshaft.

7.3 Physical model

The pressure in a cylinder of a diesel engine is given [36] by

$$p(\phi) = \frac{(\nu-1)}{V(\phi)}\left[\frac{\int_{\phi_{BB}}^{\phi} Q'(\psi)V(\psi)^{\nu-1}d\psi}{V(\psi)^{\nu-1}} + \left(\frac{V_0}{V(\phi)}\right)^{(\nu-1)} U_0\right], \tag{7.1}$$

where

$$Q(\phi) = Q_{total}\left[1 - e^{-a\left(\frac{\phi-\phi_{BB}}{\phi_{BD}}\right)^{m+1}}\right], \tag{7.2}$$

$$Q'(\phi) = \frac{dQ}{d\phi} = a(m+1)Q_{total}\left(\frac{(\phi-\phi_{BB})^m}{\phi_{BD}^{m+1}}\right)\left[e^{-a\left(\frac{\phi-\phi_{BB}}{\phi_{BD}}\right)^{m+1}}\right], \tag{7.3}$$

$$U_0 = \frac{p_a V_0}{\nu - 1}. \tag{7.4}$$

The Wiebe function is given by (7.2). Now we express the symbols in the above equations in terms of physical quantities.

TABLE 7.1
Measured quantities

ϕ	Crank angle
$p(\phi)$	Cylinder pressure

TABLE 7.2
Computed/Known quantities

$V(\phi)$	Cylinder volume at a crank angle ϕ
V_0	Cylinder volume at the crank angle 0^o
U_0	Internal energy at the crank angle 0^o
ν	Gas constant
a, m	Wiebe function constants
p_a	Atmospheric pressure

TABLE 7.3
Unknown parameters

ϕ_{BB}	Crank angle at the start of ignition
Q_{total}	Total amount of heat release due to combustion
$Q(\phi)$	Heat release due to combustion until a crank angle ϕ
ϕ_{BD}	Combustion duration

Quantities and parameters are considered in the international system (SI) of units. The goal is to estimate the combustion parameters ϕ_{BB}, Q_{total} and ϕ_{BD} from known/measured quantities in TABLEs 7.1–7.2.

We find from (7.1) that

$$p(\phi) = (\nu-1)\frac{\int_{\phi_{BB}}^{\phi} Q'(\psi)V(\psi)^{\nu-1}d\psi}{V(\phi)^{\nu}} + (\nu-1)\left(\frac{V_0^{(\nu-1)}}{V(\phi)^{(\nu)}}\right)U_0. \tag{7.5}$$

This can be formulated as

$$p(\phi) = \underbrace{(\nu-1)V(\phi)^{-\nu}\int_{\phi_{BB}}^{\phi} Q'(\psi)V(\psi)^{\nu-1}d\psi}_{p_{combustion}} + \underbrace{(\nu-1)V(\phi)^{-\nu}V_0^{(\nu-1)}U_0}_{p_{compression}}. \tag{7.6}$$

From the above equation we have that

$$p_{combustion} = (\nu-1)V(\phi)^{-\nu}\int_{\phi_{BB}}^{\phi} Q'(\psi)V(\psi)^{\nu-1}d\psi \tag{7.7}$$

$$\Rightarrow \underbrace{\frac{p_{combustion}V(\phi)^{\nu}}{(\nu-1)}}_{z(\phi)} = \int_{\phi_{BB}}^{\phi} Q'(\psi)V(\psi)^{\nu-1}d\psi. \tag{7.8}$$

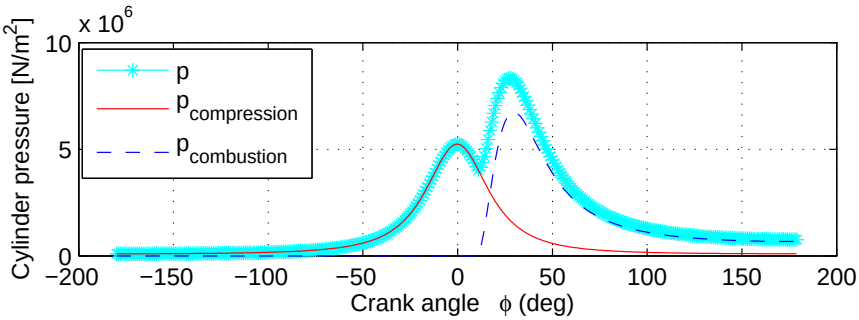

FIGURE 7.2 Plots of cylinder pressures

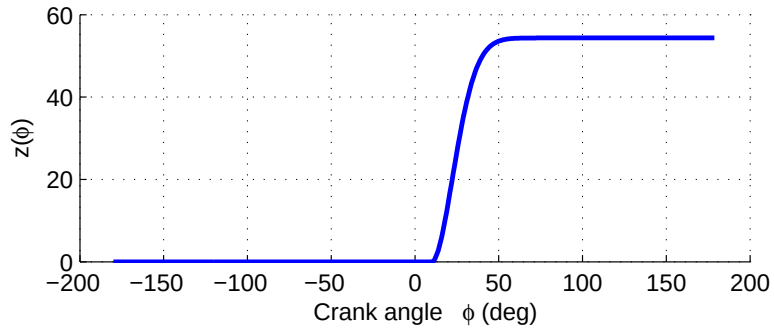

FIGURE 7.3 Plot of z

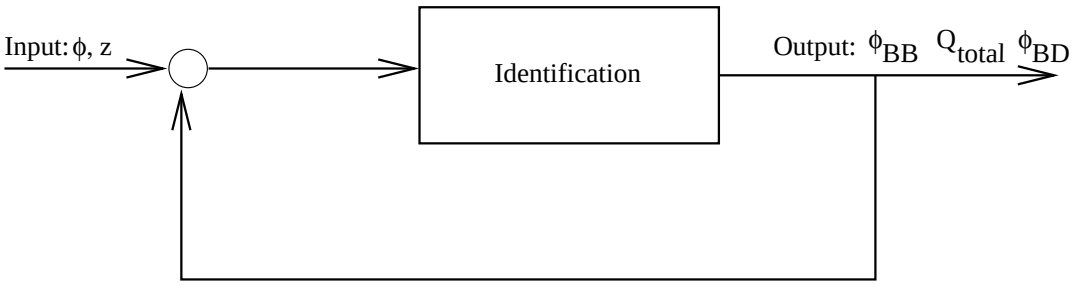

FIGURE 7.4 Control model for combustion parameters ϕ_{BB}, Q_{total} and ϕ_{BD}

FIGURE 7.2 displays the plots of the pressures $p, p_{compression}, p_{combustion}$ versus ϕ.

FIGURE 7.3 shows the plot of z versus ϕ. We use the following data for these figures.

$$\begin{aligned} \phi_{BB} &= 10^o; \quad & \phi_{BD} &= 50^o; \\ a &= 6.908; & m &= 0.8; \\ p_0 &= 1.0e5; & Q_{total} &= 2000\ J. \end{aligned}$$

FIGURE 7.4 shows the control model for combustion parameters. Measurements of pressures and crank angles are passed to the model and $V(\phi)$ is computed [36]. With these values, parameters ϕ_{BB}, Q_{total} and ϕ_{BD} can be identified by a nonlinear least squares method which we will discuss in the next section.

7.4 Proposed models

The left hand side

$$z(\phi) = \frac{p_{combustion} V(\phi)^\nu}{(\nu - 1)}$$

of (7.8) can be computed from the measurements of ϕ and $p_{combustion}$. Now we rewrite the right hand side of (7.8) using (7.3) in the form:

$$z1(x, \phi) = \quad 0 \quad \text{if } \phi < x_0, \tag{7.9}$$

$$z1(x, \phi) = \quad \int_{x_0}^{\phi} Q1'(x, \psi) V(\psi)^{\nu-1} d\psi \quad \text{if otherwise ,} \tag{7.10}$$

with

$$Q1'(x, \phi) = a(m+1)x_1 \left(\frac{(\phi - x_0)^m}{x_2^{m+1}} \right) e^{-a\left(\frac{\phi - x_0}{x_2}\right)^{m+1}} \tag{7.11}$$

and $x = [\underbrace{\phi_{BB}}_{x_0}, \underbrace{Q_{total}}_{x_1}, \underbrace{\phi_{BD}}_{x_2}]^T$.

The function $z1$ is very expensive to compute by a numerical integration and an analytical integration is not available. So we want to replace this function by some other functions which are cheap in computation and have analytical derivatives so that we can compute $Q1'$ immediately. I propose the following functions $z2, z3$, and $z4$ as potential alternatives for $z1$.

- $$z2(x, \phi) = \quad 0 \quad \text{if } \phi < x_0, \tag{7.12}$$

  $$z2(x, \phi) = \quad x_1(1 - e^{-x_2(\phi - x_0)}) \quad \text{if otherwise.} \tag{7.13}$$

- $$z3(x, \phi) = \quad 0 \quad \text{if } \phi < x_0, \tag{7.14}$$

  $$z3(x, \phi) = \quad x_1 tanh(x_2(\phi - x_0)) \quad \text{if otherwise.} \tag{7.15}$$

- $$z4(x,\phi) = x_1 tanh(x_2\phi - x_0) + x_1 \tag{7.16}$$

Here $x = [x_0, x_1, x_2]^T$.

Assume that combustion takes place in the cylinder from $\phi = \phi_{min}$ to $\phi = \phi_{max}$. To investigate the above functions, we generate data $z(\phi)$ in the combustion domain, typically, from $\phi_{min} = -10^o$ to $\phi_{max} = 100^o$, at each $\delta\phi = 6^o$ interval. We try to fit the above functions with these data. This gives a nonlinear least squares problem which residual takes the form

$$F_i(x) = z(\phi_{min} + i\delta\phi) - zp(x, \phi_{min} + i\delta\phi), \quad i = 0, \cdots, m-1, \tag{7.17}$$

with $p = 1, \cdots, 4$. Here we have 3 unknown parameters x_0, x_1 and x_2 with m nonlinear equations where m can be given by

$$m = \lceil \frac{\phi_{max} - \phi_{min}}{\delta\phi} \rceil; \tag{7.18}$$

Once we solve the nonlinear least squares with residual (7.17) for x, Qp' for $p = 2, \cdots, 4$ can be computed by the following formula derived from (7.8).

$$Qp' = \frac{\frac{\partial zp}{\partial \phi}}{V(\phi)^{\nu-1}}, \quad p = 2, \cdots, 4. \tag{7.19}$$

The derivatives $\frac{\partial zp}{\partial \phi}$ are given below.

- $$\frac{\partial z2(x,\phi)}{\partial \phi} = 0 \quad \text{if } \phi < x_0, \tag{7.20}$$
$$\frac{\partial z2(x,\phi)}{\partial \phi} = x_1 x_2 e^{-x_2(\phi - x_0)} \quad \text{if otherwise.} \tag{7.21}$$

- $$\frac{\partial z3(x,\phi)}{\partial \phi} = 0 \quad \text{if } \phi < x_0, \tag{7.22}$$
$$\frac{\partial z3(x,\phi)}{\partial \phi} = x_1 x_2 sech^2(x_2(\phi - x_0)) \quad \text{if otherwise.} \tag{7.23}$$

- $$\frac{\partial z4(x,\phi)}{\partial \phi} = x_1 x_2 sech^2(x_2\phi - x_0) \tag{7.24}$$

7.5 Results and conclusion

The Gauss-Newton method with trust-region (see chap. 4 and 5 for details) was used to solve the nonlinear least squares with residual (7.17). Jacobian matrices for the method were computed analytically. All the data of $z(\phi)$ from $\phi_{min} = -10^o$ to $\phi_{max} = 92^o$ at each $\delta\phi = 6^o$ interval were generated using (7.8). Here we took the total amount of heat release $Q_{total} = 2000\ J$ and the combustion duration $\phi_{BD} = 50^o$. Wiebe function constants were taken as $a = 6.908,\ m = 0.8$. Data sets were produced using $\phi_{BB} = -10^o, 0^o, 10^o, 20^o, 50^o$. The test was performed by a GCC compiler of version 3.3.1 under a Linux operating system, and executed on a Pentium 4 processor with 2.40 GHz. The test results are presented in TABLEs 7.4–7.5 and in FIGUREs 7.5–7.7.

TABLE 7.4
Solutions of nonlinear least squares for different functions

Index	ϕ_{BB}	Initial value				Iterations			
		z1	z2	z3	z4	z1	z2	z3	z4
1	-10	[0, 1000, 50]	[0, 40, 3]	[0, 30, 5]	[0, 30, 5]	6	6	6	7
2	0					1	9	7	7
3	10					7	10	7	8
4	20					7	11	11	9
5	50					20	20	20	20
6	-10	[1, 1000, 50]	[1, 40, 3]	[1, 30, 5]	[1, 30, 5]	10	11	14	20
7	0					11	10	20	11
8	10					9	11	3	9
9	20					11	8	11	8
10	50					6	6	7	9

In TABLE 7.4, first six columns provide the test run index, the start of ignition ϕ_{BB} of $z(\phi)$, and the initial points of x of least squares problems with residual (7.17) for the functions $z1, \cdots, z4$. The next four columns give the numbers of iterations required by the least squares method.

In TABLE 7.5, the first column contains the test run index, the next four columns present the 2-norms of residuals, and the last four columns report the computational time (in millisecond) of the method for each function. We note that the least squares method for $z1$ is the most expensive in computation, but gives the smallest residual norms at the solutions. On the other hand, the method for $z2$ requires the lowest computational time, but produces moderate residual norms at the solutions. Remarkably, the computational time for $z2$, $z3$ or $z4$ is substantially lower than that for $z1$. The reason is that the evaluation of $z1$ requires an expensive numerical integration. But, $zp,\ p = 2, \cdots, 4$ can be evaluated without much expense. Both the computational time and residual norms for $z4$ are higher than that for $z2$ and $z3$. This clearly demonstrates that $z2, z3$ are suitable alternatives for $z1$ in real time application.

TABLE 7.5
Solutions of nonlinear least squares for different functions

Index	Residual norm				Computational time			
	z1	z2	z3	z4	z1	z2	z3	z4
1	≈ 0.0	5.8	2.459	3.22	58	0.24	0.36	0.41
2	≈ 0.0	4.21	2.37	3.57	5	0.33	0.38	0.42
3	≈ 0.0	4.48	2.37	3.79	65	0.36	0.38	0.48
4	≈ 0.0	7.36	3.81	4.29	80	0.37	0.55	0.55
5	95.2	93.70	96.43	144.26	300	0.74	1.02	1.08
6	≈ 0.0	5.80037	2.46	60.98	150	0.37	0.74	1.01
7	≈ 0.0	9.0475	2.37	3.57	133	0.33	0.98	0.66
8	≈ 0.0	4.4837	3.36	3.79	86	0.36	0.47	0.53
9	≈ 0.0	7.3613	3.81	4.29	62	0.25	0.5	0.48
10	≈ 0.0	4.569	2.86	5.29	17	0.16	0.24	0.52

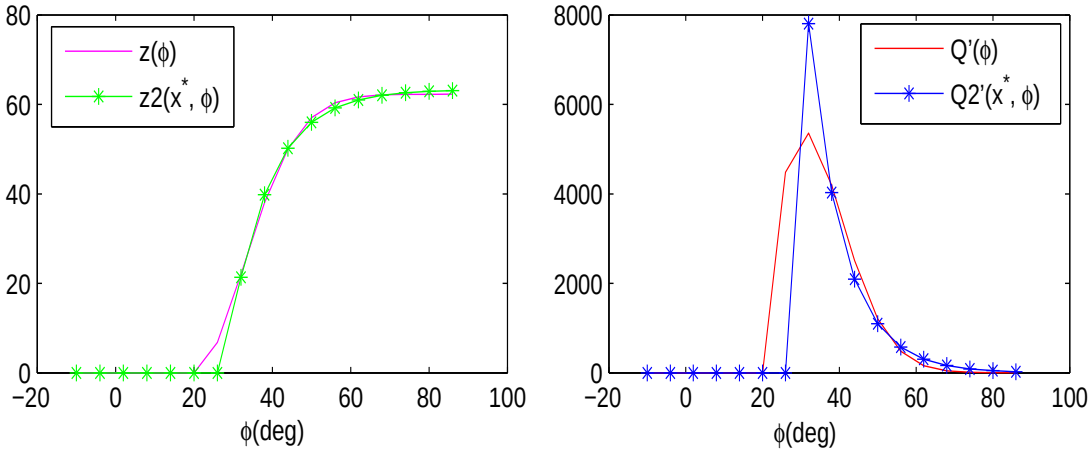

FIGURE 7.5 Nonlinear fit of $z2$ with $z(\phi)$ for the data at index 9 of TABLE 7.4.

Notice that residual norms depend on the number of equations and the magnitude of each residual component. In practice, the residual norm at a solution does not become zero due to experimental errors. Therefore, validity of our models is checked by observing how well the model functions can follow the trajectory of the Wiebe function.

FIGUREs 7.5–7.7 show the plots of $z2(x^*, \phi)$, $z3(x^*, \phi)$ and $z4(x^*, \phi)$ with $z(\phi)$. The figures also display the plots of $Q2'(x^*, \phi)$, $Q3'(x^*, \phi)$ and $Q4'(x^*, \phi)$ with $Q(\phi)$. Here x^* indicates the solution of the least squares problem with residual (7.17). We observe that $z3$ gives a better fit than $z2$. However, the computational time for $z3$ is a little higher than that for $z2$. This can also be checked from TABLE 7.5.

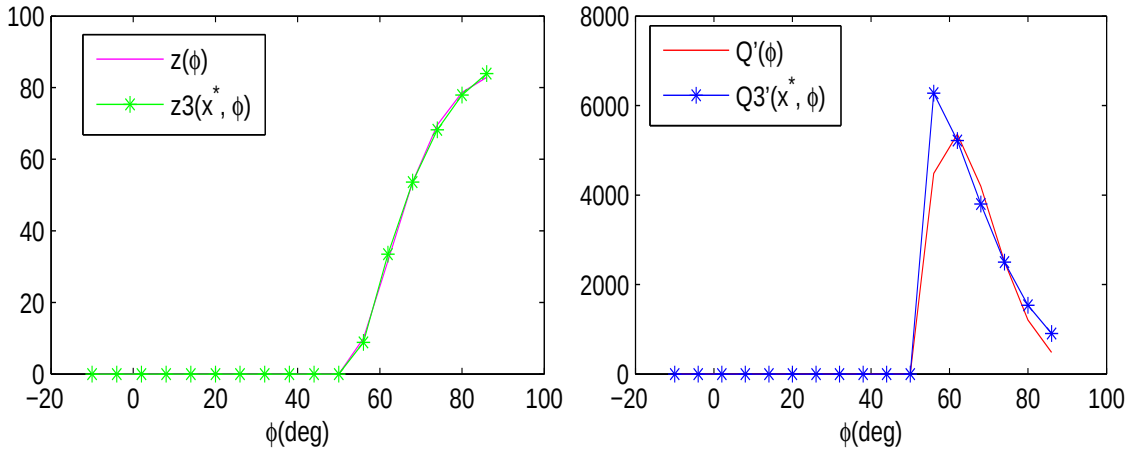

FIGURE 7.6 Nonlinear fit of $z3$ with $z(\phi)$ for the data at index 10 of TABLE 7.4.

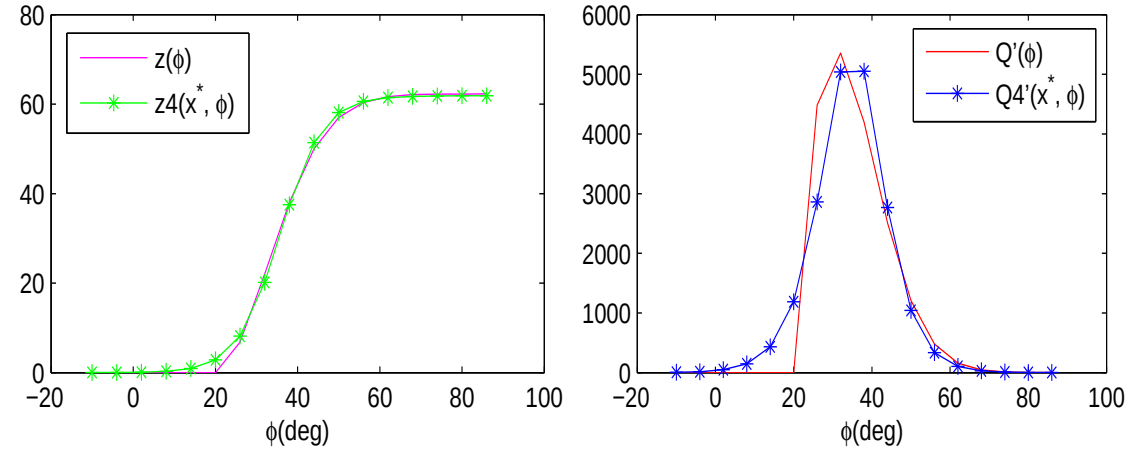

FIGURE 7.7 Nonlinear fit of $z4$ with $z(\phi)$ for the data at index 4 of TABLE 7.4.

Chapter 8

Identification of geometric tolerances in a sensor wheel

8.1 Introduction

Measurement of the speed of combustion engines is carried out by sensor wheels mounted on the crankshaft. Geometric tolerances of a sensor wheel cause systematic deviations that reduce the quality of engine speed sensing. To get the corrected speed, we have to determine the deviations due to geometric tolerances.

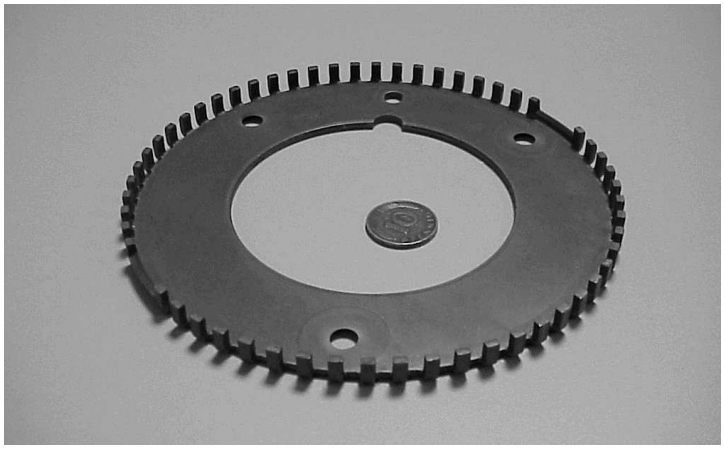

FIGURE 8.1 A 60-2-2 sensor wheel for diesel engines with 56 teeth and 2 gaps. This figure is reproduced by the courtesy of IAV GmbH, Germany

Two parameters are very important in combustion engine control systems: the crank angle at the start of fuel injection and the amount of fuel to inject. To estimate these parameters, a modern diesel fuel injection system (common rail and pump-nozzle-units) requires engine speeds, crank angles, load etc. Engine speeds and crank angles can be measured by a sensor wheel.

FIGURE 8.1 shows a 60-2-2 sensor wheel for diesel engines with 56 teeth with two gaps. The wheel mounted on the crankshaft rotates with it. The wheel is fitted with

teeth sensible by a stationary magnet which field changes with the movement of teeth. The change is sensed by the voltage generated in a coil of wire in the magnetic field.

A controller receives each voltage pulse from the sensor and records its time of arrival $T_0, T1, \cdots$. With this information, the period $\Delta T_i = T_{i+1} - T_i$ can be determined where the index $i = 0, 1, \cdots$ indicates the tooth number. The speed $\dot{\phi}_e$ can be measured by

$$\dot{\phi}_e = \frac{\Delta\phi_e}{\Delta T_i}.$$

The teeth are evenly spaced except for missing teeth in the gap. The voltage signal is interpolated over two missing teeth in each gap. The sensor wheel in FIGURE 8.1 has 60 teeth (including 4 missing teeth), and therefore $\Delta\phi_e = 360^0/60 = 6^0$. The distance between two successive teeth should be constant. However, geometric tolerances in distances between two successive teeth of sensor wheels lead to systematic deviations of the actual engine speed. Since geometric tolerances are related to these deviations by some simple formula, we will investigate the method in [15] to compute the deviations by different variations of Newton's method.

8.2 Physical model

Identification of geometric tolerances in sensor wheels is based on the balance of the kinetic energy of the crankshaft [15]. The kinetic energy can be modeled by the energy E_0 due to the engine speed and the energy E_1 due to the load:

$$E_{kin} = E_0 + E_1.$$

The kinetic energy is related to the actual engine speed by the equation

$$E_{kin} = \frac{1}{2}\Theta\dot{\phi}^2,$$

where Θ is the moment of inertia and $\dot{\phi}$ is the actual engine speed. Combining above equations together we have

$$\frac{1}{2}\Theta\dot{\phi}^2 = E_0 + E_1.$$

Due to the sensor wheel deviation, we obtain the deviated engine speed ϕ_e (which is the measured speed). The actual engine speed $\dot{\phi}$ can be given by

$$\dot{\phi}_e = \dot{\phi}(1 - \delta), \tag{8.1}$$

where δ is the speed correction due to sensor wheel deviation. Using this relation to the previous equation we get

$$\frac{1}{2}\Theta\frac{\dot{\phi}_e^2}{(1-\delta)^2} = E_0 + E_1.$$

This equation can be set up for each tooth i of the sensor wheel:

$$\frac{1}{2}\Theta_i \frac{\dot{\phi}_{e,i}^2}{(1-\delta_i)^2} = E_{0,i} + E_{1,i}, \tag{8.2}$$

where the index $i = 0, \cdots, n-1$ indicates the tooth number. Notice that the tooth position n corresponds again to the tooth position 0. We suppose that the load is constant during measurement of the engine speed.

Computing Θ

The moment of inertia Θ_i depends on the crank angle. It can be calculated by the rotating and oscillating masses of the engine. In practice, the stiff part of the moment of inertia cannot be accurately determined. Therefore, an offset Θ_{offset} is applied to the estimated moment of inertia $\Theta_{1st,i}$. Hence,

$$\Theta_i = \Theta_{offset} + \Theta_{1st,i}. \tag{8.3}$$

Computing E_0

The energy E_0 due to the engine speed changes with the friction, load, etc. The speed correction δ_0 effects on E_0 which must be taken into account. At $i = 0$, the energy E_0 due to the engine speed is given by

$$E_{0,0} = \frac{1}{2}\Theta_0 \frac{\dot{\phi}_{e,0}^2}{(1-\delta_0)^2}. \tag{8.4}$$

Similarly, at $i = n$ we have

$$E_{0,n} = \frac{1}{2}\Theta_0 \frac{\dot{\phi}_{e,n}^2}{(1-\delta_0)^2}. \tag{8.5}$$

During a crankshaft revolution, the energy due to the engine speed has approximately a linear variation. Therefore the energy $E_{0,i}$ between $E_{0,0}$ and $E_{0,n}$ can be computed by the equation

$$E_{0,i} = E_{0,0} + \frac{E_{0,n} - E_{0,0}}{n} i, \quad \text{with } i = 0, \cdots, n-1. \tag{8.6}$$

Computing E_1

The energy $E_{1,i}$ due to the load is stored during the compression stroke of the engine. It varies linearly with the load:

$$E_{1,i} = p_{load} G_{1,i}. \tag{8.7}$$

The values of $G_{1,i}$ are computed by complicated formulas which include the rules of the crankshaft kinematics and polytrope compression [15].

The goal of the identification is to estimate the unknown parameters using the known/measured set of data, and determine the actual engine speed $\dot{\phi}$. The known and unknown quantities/parameters are listed in the following table.

Known/measured quantities	Unknown parameters
$\Theta_{1st,i},\ i = 0, \cdots, n-1$	Θ_{offset}
$p_{load},\ G_{1,i},\ E_{1,i},\ i = 0, \cdots, n-1$	$\delta_i,\ i = 0, \cdots, n-1$
$\dot{\phi}_{e,i},\ i = 0, \cdots, n$	

8.3 Mathematical formulation

From (8.2), the nonlinear residual is given by

$$F_i = E_{0,i} + E_{1,i} - \frac{1}{2}(\Theta_{offset} + \Theta_{1st,i})\frac{\dot{\phi}_{e,i}^2}{(1-\delta_i)^2}, \tag{8.8}$$

with $i = 0, \cdots, n-1$ which has the unknown parameters $\delta_0, \cdots, \delta_{n-1}$ and Θ_{offset}. This system consisting of n residual components and $n+1$ unknown parameters requires an additional equation for its solution. Now we give the $(n+1)$th necessary equation which implies that the mean sensor wheel deviation vanishes:

$$F_n = \sum_{i=0}^{n-1} \delta_i. \tag{8.9}$$

Hence we have the following nonlinear system of equations

$$F(x) = 0, \tag{8.10}$$

with

$$x = [\delta_0, \delta_1, \cdots \delta_{n-1}, \Theta_{offset}]^T, \tag{8.11}$$

and

$$F = [F_0, F_1, \cdots, F_n]^T.$$

This system of nonlinear equations is solved for x and hence the corrected engine speed can be obtained using (8.1).

8.4 Results and conclusion

The system (8.10) was solved by Matlab in [15]. I solved the problem by different variants of Newton's method and analyzed the convergence and computational time. Based on this result, we want to choose an appropriate variant to solve the system in the real time operation.

Several variants of Newton's method were used to solve the nonlinear system (8.10) for δ and Θ_{offset}. All the data of $\dot{\phi}_e$, p_{load}, G_1 and Θ_{1st} from the sensor wheel with $n = 60$

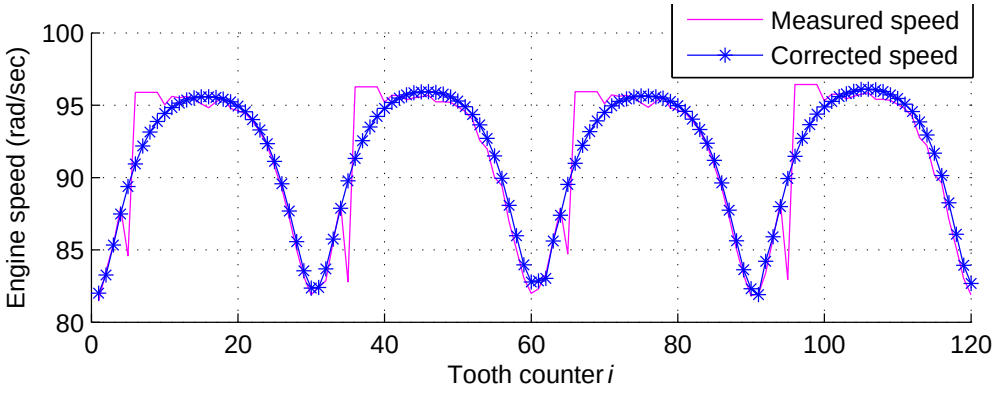

FIGURE 8.2 Measured and corrected engine speeds.

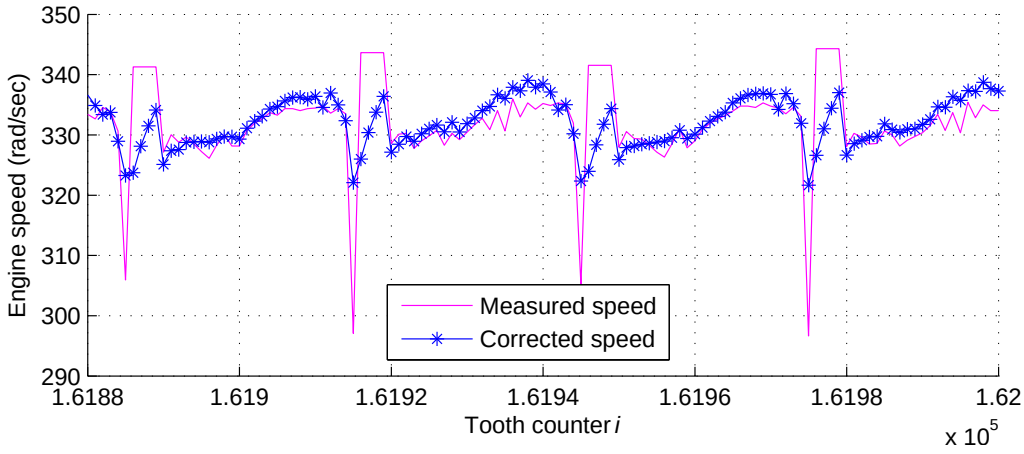

FIGURE 8.3 Measured and corrected engine speeds.

teeth were given. The Jacobian matrices were computed using center differences (5.1). The test was performed by a GCC compiler of version 3.3.1 under a Linux operating system, and executed on a Pentium 4 processor with 2.40 GHz. We obtained the corrected engine speed from δ and $\dot{\phi}_e$ using (8.1). The test results are presented in FIGUREs 8.2–8.4.

In these figures, measured and corrected engine speeds (computed from a converged solution) are plotted against the tooth counter ($i = 0, \cdots, 119$). In FIGUREs 8.2 and 8.4, the solution δ computed from the current revolution (with tooth counters $i = 0$ to 59) were used to correct the measured speeds $\dot{\phi}_e$ of the next revolution (with tooth counters $i = 60$ to 119). FIGURE 8.4 shows good results in estimating the corrected speed during the revolutions. However, In FIGURE 8.4 we observe a very good result in estimating the corrected speed during the current revolution, while the result deteriorates a little during the next revolution. The cause of the deterioration is the imperfection of computing E_1 and the effect of the crankshaft torsional vibration. The torsional vibration increases with higher engine speed. Therefore, FIGURE 8.3 shows very bad results for the revolutions in that the solution δ computed for FIGURE 8.2 (with tooth counters $i = 0$ to 59) were

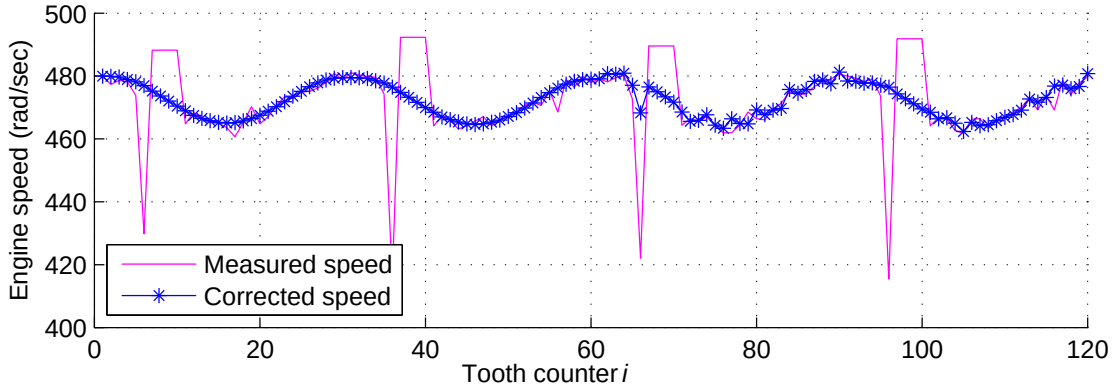

FIGURE 8.4 Measured and corrected engine speeds.

used to correct the measured speeds with tooth counters $i = 161880$ to 162000.

In the real time operation, computation for the current revolution c_k may take more time than the period δT_{k+1} of next revolution c_{k+1}. In such a case, computation continues during the next few revolutions $c_{k+1}, c_{k+1}, \cdots c_{k+p}$, and the previous solution x_{k-1} is set for these revolutions. When computation for the current revolution c_k ends, computation for the revolution c_{k+p+1} starts. This process is set to continue in the real time operation. As a demonstration, we conducted a numerical test which are shown by FIGUREs 8.5–8.7.

From FIGUREs 8.5–8.7, we assume that the engine speed is 350 radian/sec. Therefore, for 1 revolution it takes $(2\pi/350) * 1000 \approx 17$ ms (millisecond). Assume that computation takes 60 ms in a real time operation, and hence $p = 1 + 60/17 \approx 4$, that is, results of computation can be obtained after 5 revolutions. For FIGURE 8.5, we used the data of the 1st revolution was used to obtain δ to correct the measured speeds of the 6th revolution. FIGURE 8.6 shows the measured and corrected speeds where the data for the 6th revolution was used to recalculate δ to correct the measured speeds of the 11th revolution. Note that previous δ was used to correct the measured speeds of 6th to 10th revolutions. For FIGURE 8.7, the data for the 11th revolution was used to estimate δ again to correct the speeds of the 16th revolution. Previous δ was used to correct the speeds of 10th to 15th revolutions. This procedure continues in the real time operation.

Results from the different variants of Newton's method using the measured speeds shown in FIGURE 8.4 are listed in TABLE 8.1. In this table, the first four columns provide the initial point, the variant of Newton's method, the number of iterations and the 2-norms of residuals. Columns 5, 6 and 7 give us whether convergence to a true solution was achieved and the cause of failure and the required time in millisecond. "Convergence" to a solution is discussed in section 4.5. Since a nonlinear system of equations may have more than one solution, the "true solution" is meant a reasonable and practically useful solution which was verified by the criteria

- A true solution must satisfy its bound constraints. The deviations δ should be between the bounds of -0.2 and +0.2, and $\Theta_{offset} > 0$ to be physically meaningful.
- The corrected engine speed from a true solution must fit the measured speed.

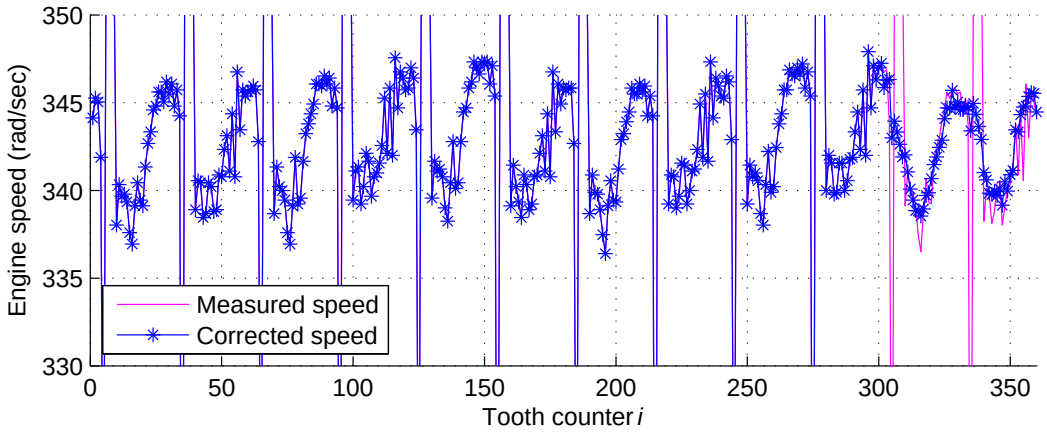

FIGURE 8.5 Measured and corrected engine speeds.

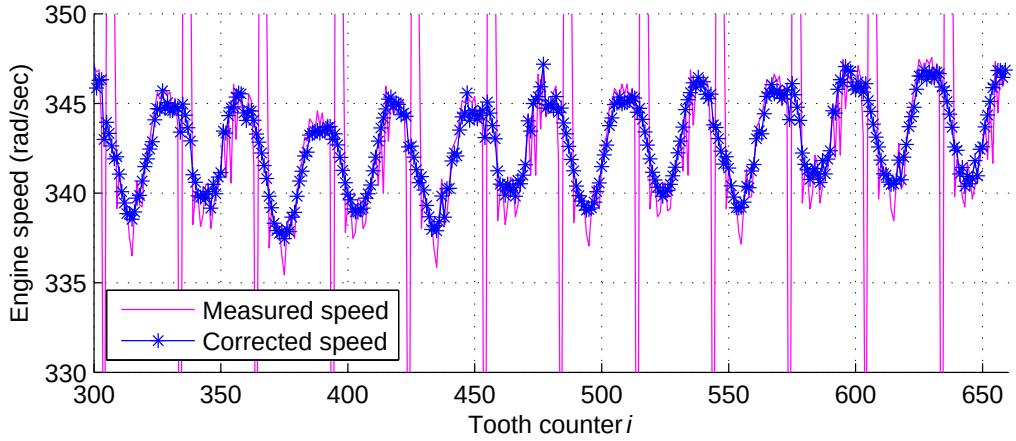

FIGURE 8.6 Measured and corrected engine speeds.

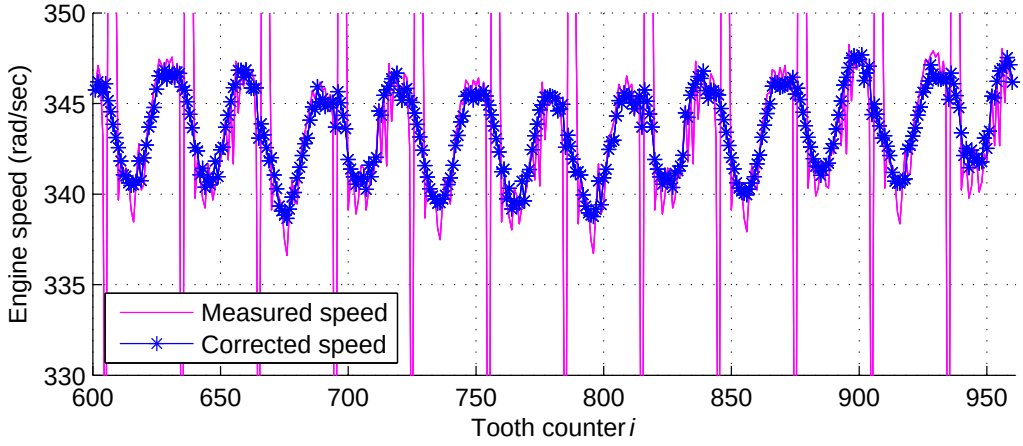

FIGURE 8.7 Measured and corrected engine speeds.

TABLE 8.1
Solutions by different variants of Newton's method

Initial point	Method	Iterations	Norm of residuals	Convergence to true solution	Cause of failure	Time(ms)
0	1	5	8.85174e-09	Yes		5.7
	2	5	8.85174e-09	Yes		5.61
	3	5	8.85174e-09	Yes		6.8
0.1	1	5	0.0255291	Yes		6.45
	2	5	1.88e-9	Yes		6.67
	3	5	0.0255291	Yes		6.8
0.2	1	5	0.0106199	Yes		6.6
	2	5	6.15e-9	Yes		6.68
	3	5	0.0106199	Yes		6.82
0.5	1	6	4.9827e-06	Yes		7.5
	2	6	6.94e-13	Yes		8.1
	3	6	5.06e-6	Yes		8.5
0.7	1	6	0.000174425	Yes		7.95
	2	6	5.48e-11	Yes		8.02
	3	6	0.0016	Yes		8.5
1	1	-	-	No	Singular	10
	2	-	-	No	Singular	15.19
	3	-	-	No	$\Delta \leq \Delta_{min}$	20.2
2	1	7	0.000701749	Yes		10
	2	7	5.8e-19	Yes		9.39
	3	-	-	No	$\Delta \leq \Delta_{min}$	20.2
10	1	-	-	No	Singular	130
	2	8	1.86e-10	Yes		1.37
	3	-	-	No	$\Delta \leq \Delta_{min}$	20.2
-0.1	1	-	-	No	Singular	120
	2	-	-	No	$\Delta \leq \Delta_{min}$	1.37
	3	-	-	No	$\Delta \leq \Delta_{min}$	20.2
-0.2	1	9	0.00471062	Yes		10
	2	-	-	No	$\Delta \leq \Delta_{min}$	1.37
	3	-	-	No	$\Delta \leq \Delta_{min}$	20.2

In column 6, "singular" indicates that the Jacobian is singular and computation stops. The other case "$\Delta \leq \Delta_{min}$" implies that the current trust-region radius Δ is less or equal to $\Delta_{min} = 1.0\text{e-}6$. In this case, further computation will hardly benefit and computation terminates. Each component of initial x holds the number given in column 1. In column 2, we use the number

- 1 for Newton's method,
- 2 for Newton's method with line search, and

- 3 for Newton's method with trust-region.

Newton's method and Newton's method with line search work better than Newton's method with trust-region for this nonlinear system of equations. With negative initial points the methods do not work properly. The methods mostly succeed if the initial points are larger than or equal to zero. We notice that the methods work well when the initial points lie between 0 and 0.2.

Chapter 9

Identification of reaction parameters in an SCR catalyst

9.1 Introduction

The Selective Catalytic Reduction (SCR) catalyst uses ammonia vapor for the reduction of harmful nitrogen oxides (NO_x) emissions from a combustion engine. Ammonia (NH_3) is adsorbed in the catalyst and NO_x over the catalyst reacts with adsorbed NH_3 to form water vapor (H_2O) and nitrogen gas (N_2). Reaction parameters are required for the design of (prediction from) an SCR catalyst. They can be identified by solving a nonlinear least squares problem that is formulated using an SCR catalyst model [23, 39] and measurements of flow at entrance and exit of the catalyst.

In this chapter, we identify the reaction parameters in an SCR catalyst which is based on a 1-dimensional model using mass and energy conservation equations considering chemical reactions. These equations are discretized in space and time domains. A finite volume method with an implicit time-stepping is used to solve these equations from which the solution at exit of the catalyst is obtained [26]. A nonlinear least squares problem is formulated using the concept that the difference between this solution and the corresponding measurement data at exit should be minimized. Finally, the least squares problem is solved for reaction parameters.

9.2 SCR model

The exhaust gas containing the species of $O_2, C_2H_6, C_3H_8, CO,\ CO_2, NO, NO_2, N_2O$, H_2O, H_2, N_2, NH_3, $HNCO$ passes through the channels of an SCR catalyst (FIGURE 9.1). The part of NH_3 is adsorbed in (desorbed from) the wash-coat. Reactions take place between the adsorbed NH_3 and other species. Since the length of the channel is large with respect to its cross-section, we will assume that flow is 1-dimensional. The species are given by a vector X as

$$X = [O_2, C_2H_6, C_3H_8, CO, CO_2, NO, NO_2, N_2O, H_2O, H_2, N_2, NH_3, HNCO]^T, \quad (9.1)$$

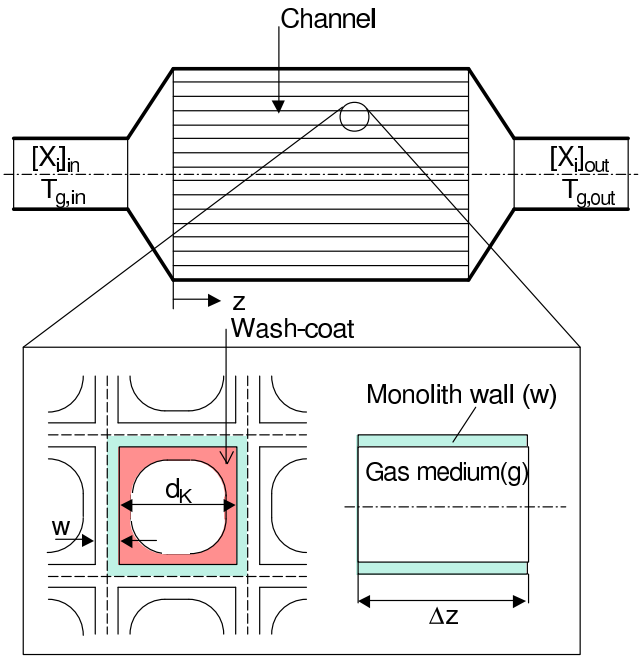

FIGURE 9.1 The figure of SCR catalyst reproduced by the courtesy of IAV GmbH, Germany

where X_i denotes the ith species.

For mass balance, we consider the mass flow through the *gas medium* and in the wash-coat (*solid medium*) separately. For energy balance, we consider the energy in the gas (*gas medium*) and in the wash-coat and monolith wall (*solid medium*) separately.

We use the subscript g (s) to indicate the associated quantity in the gas (solid) medium.

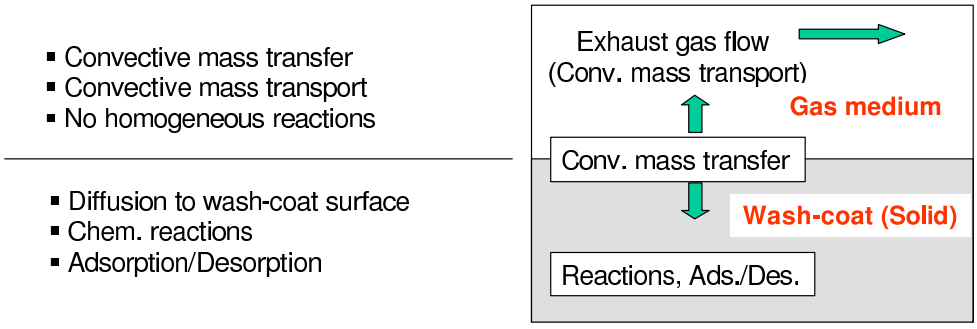

FIGURE 9.2 Mass balance in the SCR catalyst. This figure is reproduced by the courtesy of IAV GmbH, Germany

9.2.1 Mass balance

Solid medium

The governing equations of mass balance in the gas medium are given [23] by

$$\underbrace{\frac{\partial C_{i,g}}{\partial t}}_{\text{instationary}} + \underbrace{\frac{\partial(\tilde{u}C_{i,g})}{\partial z}}_{\text{convection}} = \underbrace{\frac{\partial}{\partial x}\left(D_{i,x}\frac{\partial C_{i,g}}{\partial x}\right) + \frac{\partial}{\partial y}\left(D_{i,y}\frac{\partial C_{i,g}}{\partial y}\right)}_{\text{diffusion}} \tag{9.2}$$

for $i = 1, \cdots, n$, where

- $C_{i,g}$ is the molar concentration $[mol/m^3]$ of the species i;
- $\tilde{u}$ is the molar velocity $[m/s]$ of gas flow;
- $D_{i,x}$, $D_{i,y}$ are the diffusion coefficients $[m^2/s]$ of the species i at x and y directions.

Note that we neglect convection terms in x, y directions and diffusion term in the z direction since those quantities are very small with respect to the other quantities in (9.2). To transform the above equation from 3-dimension to 1-dimension, we substitute the diffusion term by the mass flux transferred over the boundary. We now define

$$S_{i,x} := -D_{i,x}\frac{\partial C_{i,g}}{\partial x}$$

where $S_{i,x}$ is the diffusion flux $[mol/(m^2\ s)]$ for a species i in the x direction. Using Gauss divergence theorem we reveal that

$$\begin{aligned} \int_V \frac{\partial S_{i,x}}{\partial x} dV &= \int_A S_{i,x} dA \\ \Rightarrow \int_z \frac{\partial S_{i,x}}{\partial x} d_k^2 dz &= \int_z S_{i,x}(2d_k) dz, \end{aligned}$$

where d_k is the hydraulic diameter $[m]$ of the channel, V is the volume $[m^3]$ of the channel and A denotes the surface area $[m^2]$ of the channel perpendicular to x axis. This equation yields

$$\frac{\partial S_{i,x}}{\partial x} = \frac{2S_{i,x}}{d_k} = \frac{2}{d_k}\beta_i(C_{i,g} - C_{i,s}),$$

where $\beta_i(C_{i,g} - C_{i,s})$ is the mass transferred over the boundary and β_i is the mass transfer coefficient $[m/s]$ for a species i.

Similarly, using the quantities in y direction we find that

$$\frac{\partial S_{i,y}}{\partial dy} = \frac{2S_{i,y}}{d_k} = \frac{2}{d_k}\beta_i(C_{i,g} - C_{i,s}).$$

Assuming $\tilde{u}$ as a constant and using the above results in (9.2) we have that

$$\frac{\partial C_{i,g}}{\partial t} + \tilde{u}\frac{\partial C_{i,g}}{\partial z} = \frac{4}{d_k}\beta_i(C_{i,s} - C_{i,g}). \tag{9.3}$$

The quantities of this equation in physical terms are given in FIGURE 9.2.

The reaction mechanism in the SCR catalyst is presented in the following table. Note that the reaction parameters $a, \theta^0_{NH_3}, k_i, E_i$ for $i = 1, 2, \cdots, 13$ in this table are unknown of the SCR model and our aim is to identify these parameters.

TABLE 9.1
Reaction mechanism in the SCR catalyst. This table is reproduced by the courtesy of IAV GmbH, Germany.

Reaction name	Reaction equation	Reaction rate
Hydrolysis	$HNCO + H_2O \rightarrow NH_3 + CO_2$	$r_1 = k_1 e^{-\frac{E_1}{RT}} X_{HNCO} X_{H_2O}$
Adsorption/Desorption	$NH_3 \leftrightarrow (NH_3)_{ads}$	$r_2 = k_2 e^{-\frac{E_2}{RT}} X_{NH_3}(1-\theta_{NH_3})$
		$r_3 = k_3 e^{-\frac{E_3(1-a\theta_{NH_3})}{RT}} \theta_{NH_3}$
Standard SCR reaction	$4(NH_3)_{ads} + 4NO + O_2 \rightarrow 4N_2 + 6H_2O$	$r_4 = k_4 e^{-\frac{E_4}{RT}} \theta^{act}_{NH_3} X_{NO}$; $\theta^{act}_{NH_3} = \theta^0_{NH_3}\left(1 - e^{-\frac{\theta_{NH_3}}{\theta^0_{NH_3}}}\right)$
Fast SCR reaction	$4(NH_3)_{ads} + 2NO + 2NO_2 \rightarrow 4N_2 + 6H_2O$	$r_5 = k_5 e^{-\frac{E_5}{RT}} \theta^{act}_{NH_3} \frac{X_{NO} X_{NO_2}}{X_{NO}+X_{NO_2}}$
Other reaction of $NO2$	$8(NH_3)_{ads} + 6NO_2 \rightarrow 7N_2 + 12H_2O$	$r_6 = k_6 e^{-\frac{E_6}{RT}} \theta^{act}_{NH_3} X_{NO_2}$
Oxidation for $T > 380°$	$4(NH_3)_{ads} + 3O_2 \rightarrow 2N_2 + 6H_2O$	$r_7 = k_7 e^{-\frac{E_7}{RT}} \theta^{act}_{NH_3}$
	$2(NH_3)_{ads} + 2O_2 \rightarrow N_2O + 3H_2O$	$r_8 = k_8 e^{-\frac{E_8}{RT}} \theta^{act}_{NH_3}$
	$4(NH_3)_{ads} + 5O_2 \rightarrow 4NO + 6H_2O$	$r_9 = k_9 e^{-\frac{E_9}{RT}} \theta^{act}_{NH_3}$

Solid medium

The governing equations of mass balance in the solid medium (see FIGURE 9.2) have the form [23]

$$\underbrace{\frac{\partial C_{i,s}}{\partial t}}_{\text{instationary}} = \underbrace{\frac{\partial}{\partial x}\left(D_{i,x}\frac{\partial C_{i,s}}{\partial x}\right) + \frac{\partial}{\partial y}\left(D_{i,y}\frac{\partial C_{i,s}}{\partial y}\right)}_{\text{diffusion}} + \underbrace{\text{reactions}}_{\text{sources}} \tag{9.4}$$

where

- $C_{i,s}$ is the molar concentration $[mol/m^3]$ of the species i;
- d_{wc} is the wash-coat thickness $[m]$;

We set $C_{i,s} = C_g X_{i,s}$ where C_g is the total molar concentration $[mol/m^3]$ of gas and $X_{i,s}$ is the mole fraction $[mol/mol]$. Using the transformation from 3-dimension to 1-dimension as before and assuming that C_g is constant, the equation (9.4) gives

$$\frac{\partial X_{i,s}}{\partial t} = \frac{1}{d_{wc}}\beta_i(X_{i,g} - X_{i,s}) + \text{reactions}/C_g. \tag{9.5}$$

Now we write the above equation for each species.

$$\frac{\partial X_{1,s}}{\partial t} = \frac{1}{d_{wc}}\beta_1(X_{1,g} - X_{1,s}) + (-r_4 - 3r_7 - 2r_8 - 5r_9)/C_g \quad (9.6a)$$

$$\frac{\partial X_{2,s}}{\partial t} = \frac{1}{d_{wc}}\beta_2(X_{2,g} - X_{2,s}) \quad (9.6b)$$

$$\frac{\partial X_{3,s}}{\partial t} = \frac{1}{d_{wc}}\beta_3(X_{3,g} - X_{3,s}) \quad (9.6c)$$

$$\frac{\partial X_{4,s}}{\partial t} = \frac{1}{d_{wc}}\beta_4(X_{4,g} - X_{4,s}) \quad (9.6d)$$

$$\frac{\partial X_{5,s}}{\partial t} = \frac{1}{d_{wc}}\beta_5(X_{5,g} - X_{5,s}) + (r_1)/C_g \quad (9.6e)$$

$$\frac{\partial X_{6,s}}{\partial t} = \frac{1}{d_{wc}}\beta_6(X_{6,g} - X_{6,s}) + (-4r_4 - 2r_5 + 4r_9)/C_g \quad (9.6f)$$

$$\frac{\partial X_{7,s}}{\partial t} = \frac{1}{d_{wc}}\beta_7(X_{7,g} - X_{7,s}) + (-6r_6)/C_g \quad (9.6g)$$

$$\frac{\partial X_{8,s}}{\partial t} = \frac{1}{d_{wc}}\beta_8(X_{8,g} - X_{8,s}) + (-2r_5 + r_8)/C_g \quad (9.6h)$$

$$\frac{\partial X_{9,s}}{\partial t} = \frac{1}{d_{wc}}\beta_9(X_{9,g} - X_{9,s}) \quad (9.6i)$$

$$+(-r_1 + 6r_5 + 6r_5 + 12r_6 + 6r_7 + 3r_8 + 6r_9)/C_g \quad (9.6j)$$

$$\frac{\partial X_{10,s}}{\partial t} = \frac{1}{d_{wc}}\beta_{10}(X_{10,g} - X_{10,s}) \quad (9.6k)$$

$$\frac{\partial X_{11,s}}{\partial t} = \frac{1}{d_{wc}}\beta_{11}(X_{11,g} - X_{11,s}) + (4r_4 + 4r_5 + 7r_6 + 2r_7)/C_g \quad (9.6l)$$

$$\frac{\partial X_{12,s}}{\partial t} = \frac{1}{d_{wc}}\beta_{12}(X_{12,g} - X_{12,s}) + (r_1 - r_2 + r_3)/C_g \quad (9.6m)$$

$$\frac{\partial X_{13,s}}{\partial t} = \frac{1}{d_{wc}}\beta_{13}(X_{13,g} - X_{13,s}) + (-r_1)/C_g \quad (9.6n)$$

where r_p is the pth reaction rate $[mol/(m^3\ s)]$ (see TABLE 9.1). Note that NH_3 reacts only when NH_3 is adsorbed. Therefore, change in NH_3 in the solid medium depends only on the reactions of hydrolysis, adsorption and desorption.

The saturation mechanism of the wash-coat due to accumulation can be described by

$$\Gamma\frac{\partial \theta_{NH_3}}{\partial t} = r_2 - r_3 - 4r_4 - 4r_5 - 8r_6 - 4r_7 - 2r_8 - 4r_9, \quad (9.7)$$

where

- Γ is the NH_3 storage capacity of the wash-coat $[mol/m^3]$;
- θ_{NH_3} is the accumulation factor for NH_3.

9.2.2 Energy balance

Similar to mass balance, we will use 1-dimensional differential equations for energy balance (see FIGURE 9.3).

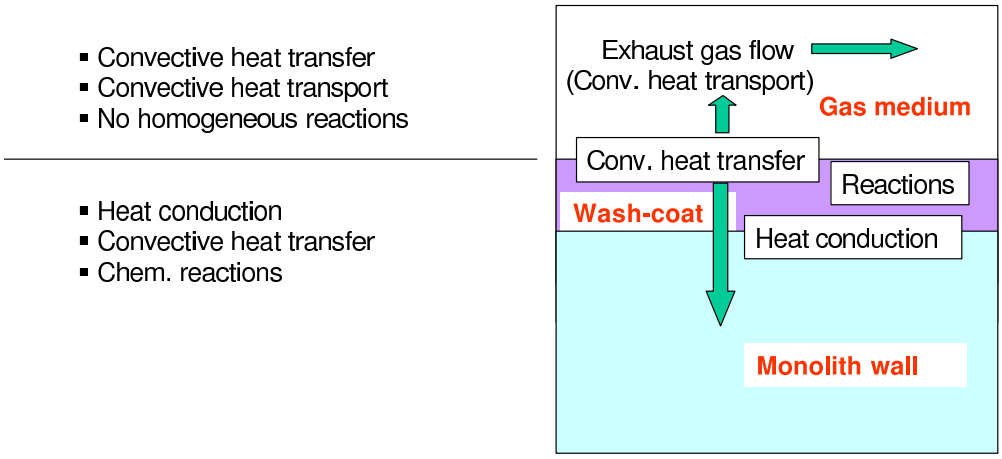

FIGURE 9.3 Energy balance in the SCR catalyst. This figure is reproduced by the courtesy of IAV GmbH, Germany

Gas medium

The governing equations of energy balance in the gas medium are given [23] by

$$\underbrace{\frac{\partial}{\partial t}\left(\rho_g c_{pg} T_g\right)}_{\text{instationary}} + \underbrace{\frac{\partial}{\partial z}\left(\rho_g u_g c_{pg} T_g\right)}_{\text{convection}} = \underbrace{\frac{\partial}{\partial z}\left(\lambda_g \frac{\partial T_g}{\partial z}\right)}_{\text{conduction}} + \underbrace{\frac{4}{d_k}\alpha(T_s - T_g)}_{\text{transfer over boundary}} \tag{9.8}$$

where

- α is the heat transfer coefficient $[W/(m^2\ K)]$;
- ρ_g is the density $[kg/m^3]$ of gas;
- $u_g \approx \tilde{u}$ is the gas velocity $[m/s]$;
- T_g is the temperature $[K]$ in the gas medium;
- c_{pg} is the specific heat capacity $[J/(kg\ K)]$ of gas;
- λ_g is the thermal conductivity $[W/(m\ K)]$ of gas.

Neglecting the instationary $\frac{\partial}{\partial t}\left(\rho_g c_{pg} T_g\right)$ and the conduction $\frac{\partial}{\partial z}\left(\lambda_g \frac{\partial T_g}{\partial z}\right)$ terms from (9.8) yields

$$\frac{\partial}{\partial z}\left(\rho_g u_g c_{pg} T_g\right) = \frac{4}{d_k}\alpha(T_s - T_g). \tag{9.9}$$

Solid medium

The governing equations of energy balance in the solid medium (wash-coat and monolith wall) are given [23] by

$$\underbrace{\frac{\partial}{\partial t}\left(\rho_s c_s T_s\right)}_{\text{instationary}} = \underbrace{\frac{\partial}{\partial z}\left(\lambda_s \frac{\partial T_s}{\partial z}\right)}_{\text{conduction}} + \underbrace{\frac{2}{w}\alpha(T_g - T_s)}_{\text{transfer over boundary}} + \underbrace{\sum_p \Delta h_p r_p}_{\text{sources}} \tag{9.10}$$

where

- ρ_s is the density $[kg/m^3]$ of solid;
- T_s is the temperature $[K]$ in the solid medium;
- c_s is the specific heat capacity $[J/(kg\ K)]$ of solid;
- λ_s is the thermal conductivity $[W/(m\ K)]$ of solid;
- w is the monolith wall thickness $[m]$;
- Δh_p is the enthalpy change $[J/mol]$ of the reaction r_p.

Note that here we use w instead $w+d_{wc}$ in the term of heat transferred over the boundary since $d_{wc} << w$.

9.2.3 Calculating constants [27]

The constants are defined as follows. The density of gas and solid are

$$\rho_g = \frac{p_a M_g}{R T_g}, \quad \rho_s = 2000.0\ kg/m^3; \tag{9.11}$$

where M_g is the total molar mass of gas, $R = 8314\ [J/(mol\ K)]$ is the universal gas constant, and $p_a = 1.013e5\ [N/m^2]$ is the atmospheric pressure. M_g is related to the molar mass M_i of each species i by

$$M_g = \sum_i M_i X_{i,g}.$$

The velocity u_g of gas is calculated by

$$u_g = \frac{\dot{m}}{\rho_g d_k^2}, \tag{9.12}$$

where $\dot{m}$ is the mass flux $[kg/s]$. Total molar concentration of gas is given by

$$C_g = \frac{p_a}{R T_g}. \tag{9.13}$$

Specific heat capacities

$$c_{pg} = (28.09 + 0.195e\text{-}2\ T_g + 0.4799e\text{-}5\ T_g^2 - 1.965e\text{-}9\ T_g^3)/M_g, \tag{9.14}$$

$$c_s = 1071 + 0.156T_s - \frac{3435e4}{T_s^2}. \tag{9.15}$$

Thermal conductivity coefficients

$$\lambda_g = 2.269e\text{-}4\ T_g^{0.832}, \tag{9.16}$$

$$\lambda_s = 1.653 - 5.06e\text{-}4\ T_s. \tag{9.17}$$

Heat transfer coefficient

$$\alpha = \frac{q\lambda_g}{d_k}, \tag{9.18}$$

where q is the Nusselt number. The Nusselt number is given by

$$q = 2.976\left(1 + 0.0095Re\ Pr\frac{d_k}{z_j}\right)^{0.45},$$

with the Reynolds number

$$Re = u_g\rho_g\frac{d_k}{\nu_d},$$

and the Prandtl number

$$Pr = \nu_d\frac{c_{pg}}{\lambda_g}.$$

Here z_j is the length of the volume element of the channel $[m]$ (see FIGURE 9.4). Dynamic viscosity

$$\nu_d = 1.0 \times 10^{-5}(1.384 + 0.00268T_g).$$

Mass transfer coefficient

$$\beta_i = \frac{S_h D_i}{d_k}, \tag{9.19}$$

with the Sherwood number $S_h = 2.89$, and

$$D_i = \left(\frac{1}{M_g}\sum_{j,\ j\neq i}\frac{M_j X_{j,g}}{E_{ij}}\right)^{-1},$$

$$E_{ij} = \frac{0.01013T_g^{1.75}\left(\frac{1}{M_i} + \frac{1}{M_j}\right)^{1/2}}{p_a\left(V_i^{1/3} + V_j^{1/3}\right)^2},$$

where V_i is the molar volume of species i. Molar mass and molar volume for each species are given in the following table.

TABLE 9.2
Molar mass and molar volume

Specie	O_2	C_2H_6	C_3H_8	CO	CO_2	NO	NO_2
M $[Kg/mol]$	32	42.081	44.09	28.01	44	30	46.01
V $[mol/m^3]$	16.6	61.4	65.34	18.9	26.0	11.7	16.7

Specie	N_2O	H_2O	H_2	N_2	NH_3	$HNCO$
M $[Kg/mol]$	44.013	18	2.016	28.02	17.031	43.0178
V $[mol/m^3]$	35.9	12.7	7.07	17.9	16.52	28.2

9.3 Solution of the SCR model

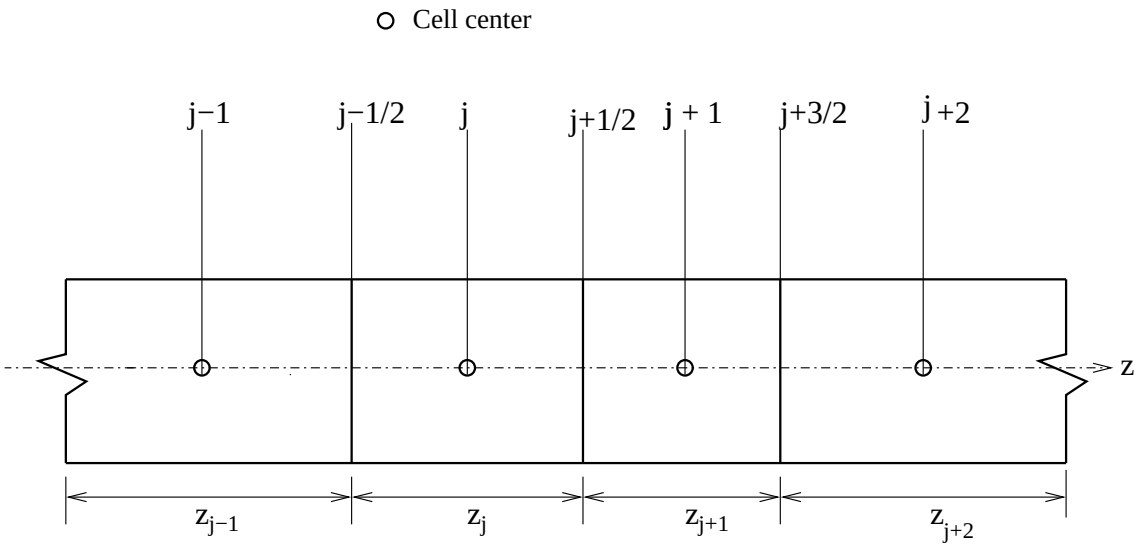

FIGURE 9.4 Discretization of the physical domain of SCR catalyst

9.3.1 Mass balance

We now perform a spatial discretization [27] for (9.3) using a cell-centered finite volume approach (see FIGURE 9.4). We set $C_{i,g} = C_g X_{i,g}$ in (9.3) and integrate this equation over a finite control volume V_j.

$$\int_{V_j} \frac{\partial C_g X_{i,g}}{\partial t} dV \quad + \quad \int_{V_j} \tilde{u} \frac{\partial C_g X_{i,g}}{\partial z} dV = \int_{V_j} \beta_i \frac{4}{d_k} C_g (X_{i,s} - X_{i,g}) dV. \qquad (9.20)$$

We check that

$$\begin{aligned}\int_{V_j} \tilde{u}\frac{\partial C_g X_{i,g}}{\partial z}dV &= \int_{A_j} \tilde{u}C_g X_{i,g}dA \\ &= \tilde{u}C_g d_k^2(X_{j+\frac{1}{2},i,g} - X_{j-\frac{1}{2},i,g}) \\ &= \dot{N}(X_{j+\frac{1}{2},i,g} - X_{j-\frac{1}{2},i,g}),\end{aligned}$$

where $\dot{N} := \tilde{u}C_g d_k^2$ is assumed as constant. Since C_g is constant, we find from (9.3) that

$$C_g\frac{dX_{j,i,g}}{dt}V_j + \dot{N}(X_{j+\frac{1}{2},i,g} - X_{j-\frac{1}{2},i,g}) = \beta_{j,i}\frac{4}{d_k}C_g(X_{j,i,s} - X_{j,i,g})V_j$$

Using upwind scheme [6], we write $X_{j,i,g} \approx X_{j+\frac{1}{2},i,g}$ and $X_{j-1,i,g} \approx X_{j-\frac{1}{2},i,g}$. Thus we conclude that

$$\frac{dX_{j,i,g}}{dt} = -\dot{N}(X_{j,i,g} - X_{j-1,i,g})/(C_g V_j) + \beta_{j,i}\frac{4}{d_k}(X_{j,i,s} - X_{j,i,g}). \tag{9.21}$$

Using the same approach for (9.6) and (9.7) we find that

$$\frac{dX_{j,1,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,1}(X_{j,1,g} - X_{j,1,s}) + (-r_4 - 3r_7 - 2r_8 - 5r_9)_j/C_g \tag{9.22a}$$

$$\frac{dX_{j,2,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,2}(X_{j,2,g} - X_{j,2,s}) \tag{9.22b}$$

$$\frac{dX_{j,3,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,3}(X_{j,3,g} - X_{j,3,s}) \tag{9.22c}$$

$$\frac{dX_{j,4,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,4}(X_{j,4,g} - X_{j,4,s}) \tag{9.22d}$$

$$\frac{dX_{j,5,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,5}(X_{j,5,g} - X_{j,5,s}) + (r_1)_j/C_g \tag{9.22e}$$

$$\frac{dX_{j,6,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,6}(X_{j,6,g} - X_{j,6,s}) + (-4r_4 - 2r_5 + 4r_9)_j/C_g \tag{9.22f}$$

$$\frac{dX_{j,7,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,7}(X_{j,7,g} - X_{j,7,s}) + (-6r_6)_j/C_g \tag{9.22g}$$

$$\frac{dX_{j,8,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,8}(X_{j,8,g} - X_{j,8,s}) + (-2r_5 + r_8)_j/C_g \tag{9.22h}$$

$$\frac{dX_{j,9,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,9}(X_{j,9,g} - X_{j,9,s}) \tag{9.22i}$$

$$+(-r_1 + 6r_4 + 6r_5 + 12r_6 + 6r_7 + 3r_8 + 6r_9)_j/C_g \tag{9.22j}$$

$$\frac{dX_{j,10,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,10}(X_{j,10,g} - X_{j,10,s}) \tag{9.22k}$$

$$\frac{dX_{j,11,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,11}(X_{j,11,g} - X_{j,11,s}) + (4r_4 + 4r_5 + 7r_6 + 2r_7)_j/C_g \tag{9.22l}$$

$$\frac{dX_{j,12,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,12}(X_{j,12,g} - X_{j,12,s}) + (r_1 - r_2 + r_3)_j/C_g \tag{9.22m}$$

$$\frac{dX_{j,13,s}}{dt} = \frac{1}{d_{wc}}\beta_{j,13}(X_{j,13,g} - X_{j,13,s}) + (-r_1)_j/C_g \tag{9.22n}$$

$$\frac{d\theta_{j,NH_3}}{dt} = (r_2 - r_3 - 4r_4 - 4r_5 - 8r_6 - 4r_7 - 2r_8 - 4r_9)_j/\Gamma, \tag{9.23}$$

Now we want to solve (9.21)–(9.23) as a coupled system. But the reaction terms in these equations make them very stiff. Therefore, the system will be solved by a *Rosenbrock method*.

Rosenbrock methods

A Rosenbrock method [33, chap. 16] is a generalization of the Runge-Kutta method for the solution of ordinary differential equations and can be used to solve a stiff set of differential equations. *Stiffness* is a phenomenon which occurs in a problem where the dependent variables change with two or more different scales of the independent variable.

We consider the problem of a stiff set of differential equations of the form

$$y' = f(y). \tag{9.24}$$

Using backward difference scheme in the above form reveals that

$$y_{p+1} = y_p + hf(y_{p+1})$$

which can be rewritten using the truncated Taylor's series approximation.

$$y_{p+1} = y_p + h\left(f(y_p) + \frac{\partial f(y_p)}{\partial y}(y_{p+1} - y_p)\right). \tag{9.25}$$

Hence we deduce that

$$y_{p+1} = y_p + h\left(1 - hJ\right)^{-1} f(y_p) \tag{9.26}$$

with the Jacobian $J = \frac{\partial f(y_p)}{\partial y}$.

The solution of (9.24) by a Rosenbrock method takes the form

$$y_{p+1} = y_p + \sum_{i=1}^{n} c_i k_i \tag{9.27}$$

with

$$k_i = hf\left(y_p + \sum_{j=1}^{i-1} w_{ij} k_j\right) + hJ \sum_{j=1}^{i} \gamma_{ij} k_j$$

which is the generalization of (9.26). The corrections k_i are obtained by solving n linear equations

$$(1 - \gamma hJ)k_i = hf\left(y_p + \sum_{j=1}^{i-1} \omega_{ij} k_j\right) + hJ \sum_{j=1}^{i-1} \gamma_{ij} k_j \tag{9.28}$$

for $i = 1, \cdots n$ where n is the order of the method. The coefficients γ, c_i, w_{ij} are fixed constants [33, chap. 16] independent of the problem. To minimize the computational cost, we introduce g_i as

$$g_i = \sum_{j=1}^{i-1} \gamma_{ij} k_j + \gamma k_i \tag{9.29}$$

and rearrange (9.28) as

$$\left(\frac{1}{\gamma h} - J\right) g_1 = f(y_j) \tag{9.30a}$$
$$\left(\frac{1}{\gamma h} - J\right) g_2 = f(y_j + a_{21} g_1) + c_{21} g_1 / h \tag{9.30b}$$
$$\left(\frac{1}{\gamma h} - J\right) g_3 = f(y_j + a_{31} g_1 + a_{32} g_2) + (c_{31} g_1 + c_{32} g_2)/h \tag{9.30c}$$
$$\left(\frac{1}{\gamma h} - J\right) g_4 = f(y_j + a_{41} g_1 + a_{42} g_2 + a_{43} g_3) + (c_{41} g_1 + c_{42} g_2 + c_{43} g_3)/h \tag{9.30d}$$
$$\vdots$$

The above linear system is solved for g_i which gives k_i. We observe that the Jacobian matrix is sparse for a large N. With the values of k_i, we can determine the solution at the next time step y_{p+1} from (9.27). The equations (9.21), (9.22) and (9.23) are organized in the fashion of (9.24) and solved by this method successfully. The Jacobian matrix is computed using center differences (5.1).

9.3.2 Energy balance

Gas medium

Using a finite volume approach [27] we integrate (9.9) over a control volume V_j (see FIGURE 9.4):

$$\int_{V_j} \frac{\partial}{\partial z} \left(\rho_g u_g c_{pg} T_g\right) dV = \int_{V_j} \frac{4}{d_k} \alpha (T_s - T_g) dV.$$

This reveals that

$$\left[\rho_g u_g d_k^2 c_{pg} T_g\right]_{j+\frac{1}{2}} - \left[\rho_g u_g d_k^2 c_{pg} T_g\right]_{j-\frac{1}{2}} = \frac{4}{d_k} \alpha_j (T_{j,s} - T_{j,g}) V_j.$$

We set $\dot{m} = \rho_g u_g d_k^2$ where $\dot{m}$ is the mass flux $[kg/s]$. Assuming that $\dot{m}$ is constant and $[c_{pg}]_j = c_{j,g}$, we find that

$$\dot{m}(c_{j+\frac{1}{2},g} T_{j+\frac{1}{2},g} - c_{j-\frac{1}{2},g} T_{j-\frac{1}{2},g}) = \frac{4}{d_k} \alpha_j \left[T_{j,s} - 0.5(T_{j+\frac{1}{2},g} + T_{j-\frac{1}{2},g})\right] V_j$$
$$\Rightarrow \left(\dot{m} c_{j+\frac{1}{2},g} + \frac{2}{d_k} \alpha_j V_j\right) T_{j+\frac{1}{2},g} = c_{j-\frac{1}{2},g} T_{j-\frac{1}{2},g} + \frac{2}{d_k} \alpha_j (2T_{j,s} - T_{j-\frac{1}{2},g}) V_j$$

Using upwind scheme [6], we have that $T_{j-1,g} \approx T_{j-\frac{1}{2},g}$, and hence,

$$\left(\dot{m}c_{j+\frac{1}{2},g} + \frac{2}{d_k}\alpha_j V_j\right) T_{j+\frac{1}{2},g} = c_{j-\frac{1}{2},g}T_{j-1,g} + \frac{2}{d_k}\alpha_j(2T_{j,s} - T_{j-1,g})V_j$$
$$\Rightarrow T_{j+\frac{1}{2},g} = \frac{c_{j-\frac{1}{2},g}T_{j-1,g} + \frac{2}{d_k}\alpha_j(2T_{j,s} - T_{j-1,g})V_j}{\dot{m}c_{j+\frac{1}{2},g} + \frac{2}{d_k}\alpha_j V_j}, \tag{9.31}$$

where

$$\begin{aligned} c_{j+\frac{1}{2},g} &= \frac{z_{j+1}c_{j,g} + z_j c_{j+1,g}}{z_{j+1} + z_j} \\ T_{j+\frac{1}{2},g} &= \frac{z_{j+1}T_{j,g} + z_j T_{j+1,g}}{z_{j+1} + z_j}. \end{aligned}$$

The equation (9.31) is the final form of the energy balance equation.

Solid medium

Integrating (9.10) over a control volume $V_{j,s}$ (see FIGURE 9.4), we have

$$\begin{aligned} \int_{V_{j,s}} \frac{\partial}{\partial t}\left(\rho_s c_s T_s\right) dV_s &= \int_{V_{j,s}} \frac{\partial}{\partial z}\left(\lambda_s \frac{\partial T_s}{\partial z}\right) dV_s + \int_{V_{j,s}} \frac{2}{w}\alpha(T_g - T_s) dV_s \\ &\quad + \int_{V_{j,s}} \sum_t \Delta h_t r_t dV_s, \end{aligned} \tag{9.32}$$

where $A_{j,s}$ is the surface area of the jth element of solid medium (wash-coat and monolith) and

$$\int_{V_{j,s}} \frac{\partial}{\partial t}\left(\rho_s c_s T_s\right) V_s = \frac{d(c_{j,s}T_{j,s})}{dt}\rho_s A_{j,s} z_j,$$

$$\begin{aligned} \int_{V_{j,s}} \frac{\partial}{\partial z}\left(\lambda_s \frac{\partial T_s}{\partial z}\right) dV_s &= \left[\lambda_s \frac{\partial T_s}{\partial z}\right]_{j+\frac{1}{2}} A_{j,s} - \left[\lambda_s \frac{\partial T_s}{\partial z}\right]_{j-\frac{1}{2}} A_{j,s} \\ &= \left(\frac{z_j\lambda_{j+1,s} + z_{j+1}\lambda_{j,s}}{z_{j+1} + z_j}\right) \frac{T_{j+1,s} - T_{j,s}}{(z_{j+1} + z_j)/2} A_{j,s} + \\ &\quad \left(\frac{z_j\lambda_{j-1,s} + z_{j-1}\lambda_{j,s}}{z_{j-1} + z_j}\right) \frac{T_{j-1,s} - T_{j,s}}{(z_{j-1} + z_j)/2} A_{j,s} \\ &= \left[\left(z_j\lambda_{j+1,s} + z_{j+1}\lambda_{j,s}\right) \frac{T_{j+1,s} - T_{j,s}}{(z_{j+1} + z_j)^2} \right. \\ &\quad \left. \left(z_{j-1}\lambda_{j,s} + z_j\lambda_{j-1,s}\right) \frac{T_{j-1,s} - T_{j,s}}{(z_{j-1} + z_j)^2}\right] w d_k, \end{aligned}$$

$$\int_{V_{j,s}} \frac{2}{w}\alpha(T_{j,g} - T_{j,s}) dV_s = \frac{2}{w}\alpha(T_g - T_s)[4(w/2)z_j d_k] = 4\alpha(T_{j,g} - T_{j,s}) z_j d_k,$$

$$\int_{V_{s,j}} \sum_t \Delta h_t r_t dV_s = \sum_t \Delta h_t r_{j,t}[4(w/2)z_j d_k] = 2w \sum_t \Delta h_t r_{j,t} z_j d_k.$$

Using the results we obtain from (9.32) that

$$\begin{aligned}
\frac{d(c_{j,s}T_{j,s})}{dt}\rho_s A_{j,s} z_j &= \underbrace{\frac{z_j\lambda_{j+1,s}+z_{j+1}\lambda_{j,s}}{(z_{j+1}+z_j)^2} w d_k}_{a_{j+1}} T_{j+1,s} \\
&\quad \underbrace{-\left(\frac{z_j\lambda_{j+1,s}+z_{j+1}\lambda_{j,s}}{(z_{j+1}+z_j)^2} w + \frac{z_j\lambda_{j-1,s}+z_{j-1}\lambda_{j,s}}{(z_{j-1}+z_j)^2} w + 4\alpha z_j\right) d_k}_{a_j} T_{j,s} \\
&+ \underbrace{\frac{z_j\lambda_{j-1,s}+z_{j-1}\lambda_{j,s}}{(z_{j-1}+z_j)^2} w d_k}_{a_{j-1}} T_{j-1,s} + \underbrace{4\alpha z_j d_k T_{j,g} + 2w\sum_t \Delta h_t r_{j,t} z_j d_k}_{b_j} \\
&= a_{j+1}T_{j+1} + a_j T_j + a_{j-1}T_{j-1} + b_j.
\end{aligned}$$

Using a semi-implicit method [6] this equation gives

$$\begin{aligned}
\frac{c_{p+1,j,s}T_{p+1,j,s} - c_{p,j,s}T_{p,j,s}}{\delta t_p}\rho_s A_{j,s} z_j &= \omega(a_{p+1,j+1}T_{p+1,j+1} + a_{p+1,j}T_{p+1,j} \\
&\quad + a_{p+1,j-1}T_{p+1,j-1} + b_{p+1,j}) \\
&+ (1-\omega)(a_{p,j+1}T_{p,j+1} + a_{p,j}T_{p,j} + a_{p,j-1}T_{p,j-1} + b_{p,j}).
\end{aligned}$$

This gives

$$\begin{aligned}
\frac{c_{p+1,j,s}T_{p+1,j,s} - c_{p,j,s}T_{p,j,s}}{\delta t_p}\rho_s A_{j,s} z_j &= \omega a_{p+1,j+1}T_{p+1,j+1} + \omega a_{p+1,j}T_{p+1,j} \\
&\quad + \omega a_{p+1,j-1}T_{p+1,j-1} + \omega b_{p+1,j} + (1-\omega)(a_{p,j+1}T_{p,j+1} \\
&\quad + a_{p,j}T_{p,j} + a_{p,j-1}T_{p,j-1} + b_{p,j}),
\end{aligned}$$

which can be formulated as

$$\tilde{a}_{p+1,j+1}T_{p+1,j+1} + \tilde{a}_{p+1,j}T_{p+1,j} + \tilde{a}_{p+1,j-1}T_{p+1,j-1} = \tilde{b}_{p+1,j}, \tag{9.33}$$

where

$$\begin{aligned}
\tilde{a}_{p+1,j+1} &= -\omega a_{p+1,j+1} = -\omega\frac{z_j\lambda_{p+1,j+1,s}+z_{j+1}\lambda_{p+1,j,s}}{(z_{j+1}+z_j)^2} w d_k, \\
\tilde{a}_{p+1,j} &= -\omega a_{p+1,j} + \frac{c_{p+1,j,s}}{\delta t_p}\rho_s A_{j,s} z_j \\
&= \omega\left[\frac{z_j\lambda_{p+1,j+1,s}+z_{j+1}\lambda_{p+1,j,s}}{(z_{j+1}+z_j)^2} w + \frac{z_j\lambda_{p+1,j-1,s}+z_{j-1}\lambda_{p+1,j,s}}{(z_{j-1}+z_j)^2} w + 4\alpha z_j\right] d_k \\
&\quad + \frac{c_{p+1,j,s}}{\delta t_p}\rho_s A_{j,s} z_j, \\
\tilde{a}_{p+1,j-1} &= -\omega a_{p+1,j-1} = -\omega\frac{z_j\lambda_{p+1,j-1,s}+z_{j-1}\lambda_{p+1,j,s}}{(z_{j-1}+zj)^2} w d_k,
\end{aligned}$$

$$
\begin{aligned}
\tilde{b}_{p+1,j} &= \omega b_{p+1,j} + (1-\omega)(a_{p,j+1}T_{p,j+1} + a_{p,j}T_{p,j} + a_{p,j-1}T_{p,j-1} + b_{p,j}) \\
&\quad + \frac{c_{p,j,s}T_{p,j,s}}{\delta t_p}\rho_s z_j A_{j,s} T_{p,j} \\
&= 4\alpha d_k z_j T_{j,g} + 2wd_k z_j \sum_t \Delta h_t r_{p+1,j,t} + 2(1-\omega)wd_k z_j \sum_t \Delta h_t r_{p,j,t} \\
&+ (1-\omega)\left\{ \frac{z_j\lambda_{p,j+1,s} + z_{j+1}\lambda_{p,j,s}}{(z_{j+1}+z_j)^2} wd_k T_{p,j+1} \right. \\
&- \left[\frac{z_j\lambda_{p,j+1,s} + z_{j+1}\lambda_{p,j,s}}{(z_{j+1}+z_j)^2} wd_k + \frac{z_j\lambda_{p,j-1,s} + z_{j-1}\lambda_{p,j,s}}{(z_{j-1}+z_j)^2} wd_k + 4\alpha d_k z_j \right] T_{p,j} \\
&+ \left. \frac{z_j\lambda_{p,j-1,s} + z_{j-1}\lambda_{p,j,s}}{(z_{j-1}+z_j)^2} wd_k T_{p,j-1} \right\} \\
&+ \frac{c_{p,j,s}T_{p,j,s}}{\delta t_p}\rho_s z_j A_{j,s} T_{p,j}.
\end{aligned}
$$

The triangular system (9.33) can be solved efficiently by Thomas Algorithm [6].

9.4 Nonlinear least squares

The goal is to identify the reaction parameters $a, \theta^0_{NH_3}, k_i, E_i, \Gamma$ for $i = 1, 2, \cdots, 13$, given in Table 9.1 and (9.23) by a nonlinear least squares method. The nonlinear residual can be formulated for each component i as

$$F_{i,p}(x) = y_{i,p} - h_i(t_p, x), \quad p = 1, \cdots, m, \tag{9.34}$$

where

- $x = [k_1, k_2, \cdots, a, E_1, E_2, \cdots, \theta^0_{NH_3}, \Gamma]^T$;
- $y_{i,p}$ is the measured output from the catalyst, i.e., $y_{i,p}$ is the vector of measured mole fractions $X_{i,g}$ of the species i at the exit of catalyst at time t_p;
- $h_i(t_p, x)$ is the output $X_{i,g}$ from the SCR model obtained by solving (9.21–9.23), (9.31) and (9.33) at time t_p.

We consider the output data of each species one by one, i.e.

Specie	1	2	$\cdots$	13
t_p	$t_1, t_2, \cdots, t_m$	$t_1, t_2, \cdots, t_m$	$\cdots$	$t_1, t_2, \cdots, t_m$
$y_{i,p}$	$y_{1,1}, y_{1,2}, \cdots, y_{1,m}$	$y_{2,1}, y_{2,2}, \cdots, y_{2,m}$	$\cdots$	$y_{13,1}, y_{13,2}, \cdots, y_{13,m}$

Note that m is the dimension of output data for each species. Now we rewrite (9.34) as follows:

$$
\begin{aligned}
F_{1,p}(x) &= y_{1,p} - h_1(t_p, x), \quad p = 1, \cdots, m && (9.35)\\
F_{2,p}(x) &= y_{2,p} - h_2(t_p, x), \quad p = 1, \cdots, m && (9.36)\\
&\vdots && (9.37)\\
F_{13,p}(x) &= y_{13,p} - h_{13}(t_p, x), \quad p = 1, \cdots, m. && (9.38)
\end{aligned}
$$

The model output $h_i(t_p, x)$ must have same sequence as in $y_{i,p}$.

To reduce computational expenses, we try to decouple the reaction system into subsystems. The parameters associated with reactions which actually take place in each subsystem are identified.

For example, if we experiment by injecting NH_3 in the exhaust gas upstream of the catalyst, then we have NH_3 and normal air O_2 and N_2 in the catalyst. Assume that temperature in the catalyst is below 380^o. Hence, we only have adsorption and desorption processes in the reaction system (see Table 9.1) which is a subsystem consisting of adsorption and desorption processes. Now the associated parameters $x = [k_2, k_3, E_2, E_3, a, \Gamma]^T$ have to be identified. In this case, the measured output only has the data of NH_3 since we have no other input (output) in (from) the reactions. Therefore, we only consider the residual associated with the species NH_3:

$$F_{12,p}(x) \quad = \quad y_{12,p} - h_{12}(t_p, x), \quad p = 1, \cdots, m. \tag{9.39}$$

9.5 Results and conclusion

TABLE 9.3
Solutions of nonlinear least squares

x_0	$\|F(x_0)\|_2$	Iterations	x^*	$\|F(x^*)\|_2$
[5.526164e4, 6.528556e5, 0.20004, 5.57712e4, 5.40567667e4, 500]	7.831e6	35	[2.15618e4, 2.93614e5, 0.265994, 0.0, 5.42941e4 551.44]	2.450e6
[0.552616e5, 6.5286e5, 0.20004, 55.7712, 5.4057e4, 500]	2.63224e6	23	[0.538291e5, 0.403161e7, 0.162761, 7.91092e-9, 0.642595e5, 453.219]	2.181e6
[0.413689e4, 2.50132e5, 0.170035, 0.0, 4.99906e4, 500]	2.46158e6	22	[0.415955e4, 2.5021e5, 0.164695, 0.0, 4.99989e4 482.063]	2.260e6

We investigated adsorption and desorption processes of ammonia NH_3 and identified the parameters $k_2, k_3, E_2, E_3, a, \Gamma$. Data of mass flux $\dot{m}$, temperature T_g, mole fraction $X_{12,g}$ of NH_3 at entrance (and exit) of the catalyst were available from measurements. The Gauss-Newton method with trust-region was used to solve the nonlinear least squares with residual (9.39). Jacobian matrices for the method were computed using center differences (5.1). The test was performed by a GCC compiler of version 3.3.1 under a Linux operating system, and executed on a Pentium 4 processor with 2.40 GHz.

Test results from the Gauss-Newton method with trust-region are presented in TABLE 9.3 and in FIGUREs 9.5–9.7. Measured and model outputs of $X_{12,g}$ were scaled by a factor of 1.0e3 to avoid numerical errors (see chap. 2 and chap. 5) in solving our least squares problem. Moreover, we used the scaling factor

$$S = diag[1.0e4,\ 1.0e5,\ 1,\ 1.0e4,\ 1.0e4,\ 5.0e2]$$

for x to obtain the values of x_0 between 1 to 100. We present the scaled results of $X_{12,g}$ in the table and figures.

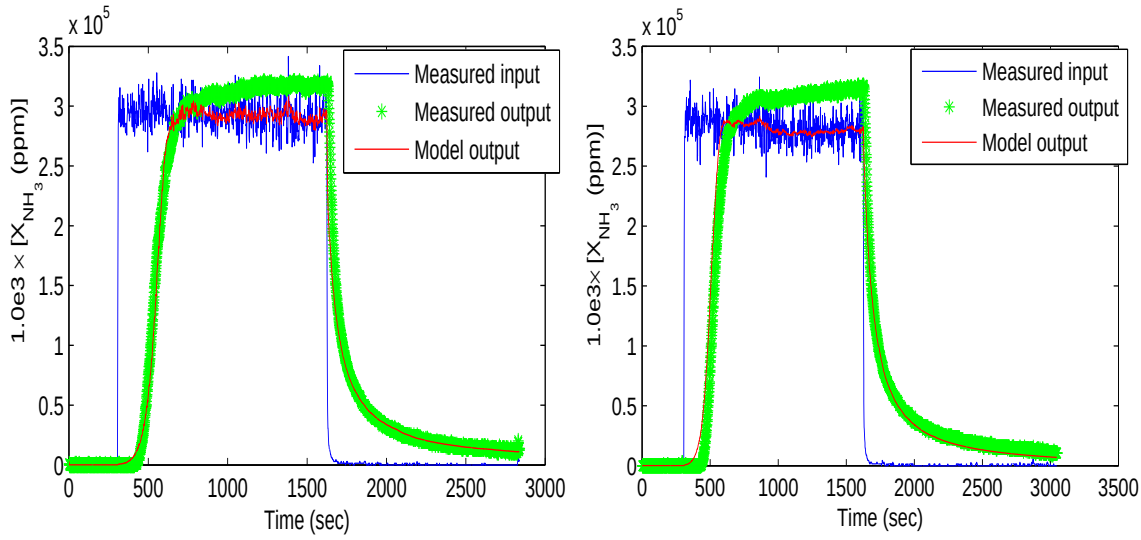

FIGURE 9.5 NH_3 adsorption and desorption for measurement 1–2

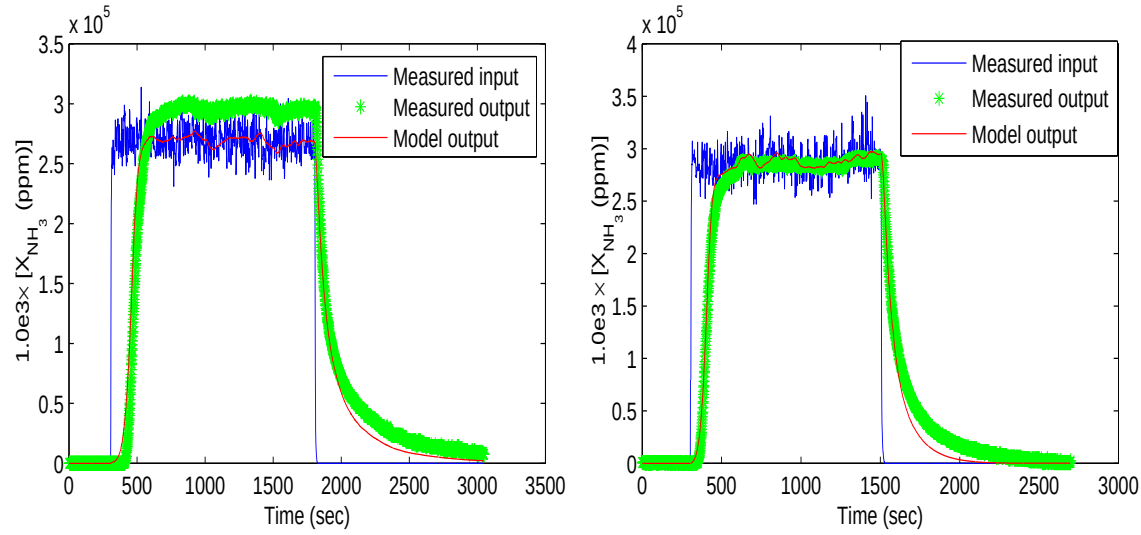

FIGURE 9.6 NH_3 adsorption and desorption for measurement 3–4

In TABLE 9.3, the first column contains the initial point x_0 which was obtained from the genetic algorithm [34] tool developed in Matlab by Müller de Vries [11] at IAV. The genetic algorithm searches approximate solutions of the residual equation (9.39) with different initial points. We have a good chance to obtain an approximate global solution using the genetic algorithm with initial points scattered through the possible solution region.

Column 2 of TABLE 9.3 presents the 2-norm of residual at x_0. The last three columns

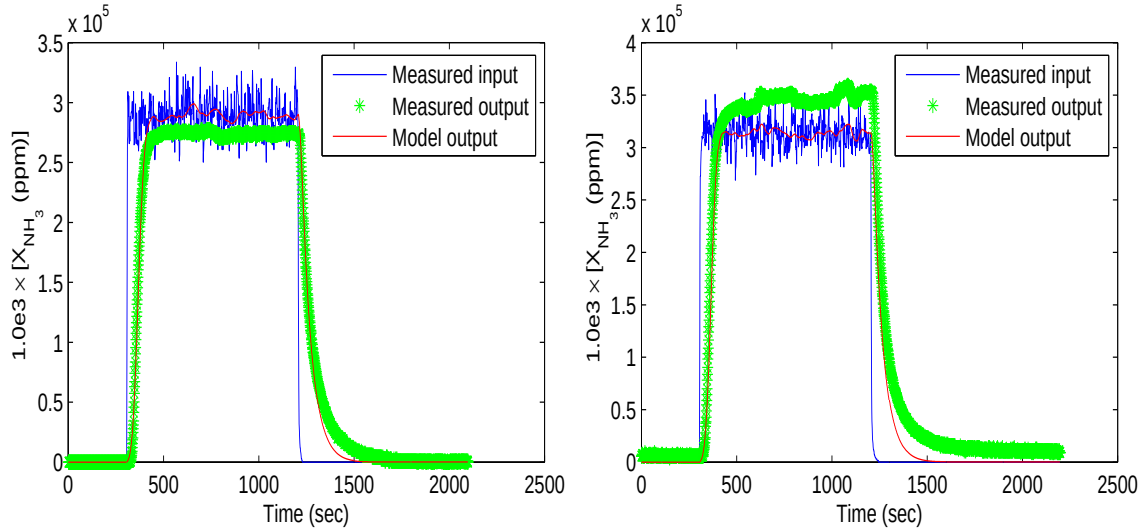

FIGURE 9.7 NH_3 adsorption and desorption for measurement 5–6

provide the number of iterations to converge, the convergent point x^* and the 2-norm of residual at x^*.

To get closer to the solution, we used the genetic algorithm tool to solve the residual equation (9.39). We took the solutions from this tool as initial points for our Gauss-Newton method with trust-region. Improvement of the results can be checked by the norm of residual in TABLE 9.3. Average computational time of our algorithm was 22.43 hours for the three solutions in the table. Notice that residual norms rely on the number of residual components, and the magnitude and scaling factor of each residual component. In practice, the residual norm at a solution does not become zero due to experimental errors. Therefore, validity of our SCR model is checked by observing how well the model output can follow the trajectory of the measured output. The solutions were quite satisfactory for the IAV.

FIGUREs 9.5–9.7 show the plots of input, output from measurements, and model output for different measurements. The model was solved using the best solution x^* from TABLE 9.3.

In the measurements, NH_3 were injected for a certain time period in the exhaust gas upstream of the catalyst at a constant temperature. The mole fractions $X_{12,g}$ in ppm (parts per million) of NH_3 at entrance and exit of the catalyst were measured. NH_3 is adsorbed in the catalyst till it is saturated. This effect can be checked by observing the trajectories of inputs and outputs (from measurements) and model outputs in the figures. After the saturation, input and output NH_3 should be equal. However, increase in measured output shows experimental errors. Model output shows better result in this case since our solution is obtained from the conservative (integral) equations of mass and energy balances. When injection of NH_3 is stopped, the output reduces steeply, but takes some time to reduce to zero since mass flux becomes very small inside the catalyst.

Further investigation on the other reaction parameters will be carried out when the

measurement data are available.

Chapter 10

Conclusion

10.1 Achieved goal

A software was developed for solving the problems of nonlinear optimization, nonlinear equations and nonlinear least squares by Newton's method and its variants (Newton's method with line search/trust-region). Test results show that Newton's method with trust-region gives the best convergence result while requiring highest number of iterations. But Newton's method for the identification problem of sensor wheels demonstrates better performance than its variants. Therefore it is unlikely that a single variant will always give the best performance for different varieties of identification problems. Key issues are the selection of initial points for the methods. Theory has little to say about such matters, but poor choices lead to poor performance.

The achievement from this dissertation can be summarized as follows.

1. A software developed in C and ASCET-C languages for solving the problems of nonlinear unconstrained optimization, nonlinear equations and nonlinear least squares by Newton's method and its variants. This software is stand-alone which does not call any external routine.

2. A new trust-region algorithm was developed for solving nonlinear system of equations and nonlinear least squares.

3. Alternative functions to the Wiebe function were formulated for modeling the combustion process in a diesel engine.

10.2 Further work

Further work can be concentrated on the following topics.

1. For a specific problem, sparsity of the Hessian/Jacobian matrix can be taken into account.

2. Quasi-Newton methods can be implemented with the line search and trust-region methods.

3. The model of sensor wheels can be improved by considering the effect of crankshaft torsional vibrations.

Bibliography

[1] *CBLAS (f2c'ed version of Basic Linear Algebra Subprograms, http://www.netlib.org/clapack/cblas.*

[2] *CLAPACK (f2c'ed version of Linear Algebra PACKage, http://www.netlib.org/clapack.*

[3] *NETLIB: A collection of mathematical software, papers, and databases, http://www.netlib.org.*

[4] *Numerical Computation Guide, http://docs.sun.com/source/806-3568/ncgTOC.html.* Sun Microsystems, Inc.

[5] *IEEE Standard for Binary Floating-Point Arithmetic: ANSI/IEEE Standard 754-1985.* Institute of Electrical and Electronics Engineers, 1985.

[6] W. W. Baumann, U. Bunge, O. Frederich, M. Schatz, and F. Thiele. *Manuskript zur Vorlesung: Finite-Volumen-Methode in der Numerischen Thermofluiddynamik.* Institute of fluid mechanics and technical acoustics, Technical University of Berlin, Germany, 2006.

[7] F. V. Berghen. *CONDOR: a constrained, non-linear, derivative-free parallel optimizer for continuous, high computing load, noisy objective functions.* PhD thesis, Faculty of Polytechnics, University of Brussels, Belgium, June 2004.

[8] U. Breymann. *C++ : Einführung und professionelle Programmierung.* Hanser, 2005.

[9] R. H. Byrd, M. Marazzi, and J. Nocedal. *On the Convergence of Newton Iterations to Non-stationary Points.* Technical report, Optimization Technology Center, Northwestern University, April 2001.

[10] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region methods.* Siam, 2000.

[11] I. M. de Vries. *Codes of genetic algorithm tool using Matlab.* MDD34, IAV GmbH, Gifhorn, Germany, July 2006.

[12] J. W. Demmel. *Applied Numerical Linear Algebra.* Siam, 1997.

[13] P. Deuflhard and A. Hohmann. *Numerische Mathematik I. Eine algorithmisch orientierte Einfhrung.* de Gruyter, 2002.

[14] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers.* Siam, 1998.

[15] H. Fehrenbach, C. Hohmann, T. Schmidt, W. Schultalbers, and H. Rasche. *Kompensation des Geberradfehlers im Fahrbetrieb. MTZ journal*, pages 588–591, Aug. 2002.

[16] N. I. M. Gould and S. Leyffer. *An introduction to algorithms for nonlinear optimization.* Computational Science and Engineering Department, Atlas Centre, Rutherford Appleton Laboratory Oxfordshire OX11 0QX, January 2006.

[17] M. T. Heath. *Scientific Computing: An Introductory Survey.* McGraw-Hill, 2002.

[18] N. J. Higham. *Accuracy and stability of Numerical algorithms.* Siam, 2002.

[19] J. C. Huang and T. Leng. *Generalized Loop-Unrolling: a Method for Program Speed-Up.* Technical report, Department of Computer Science, University of Houston.

[20] S. V. Huffel and J. Vandewalle. *Total Least Squares Problem; Computational Aspects and Analysis.* Siam, 1991.

[21] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Siam, 1996.

[22] A. Jung and U. Reddy. *Lecture note: Introduction to Computer Science.* School of Computer Science, The University of Birmingham, 2002.

[23] S. Käfer. *Trochenharnstoff-SCR-System und Betriebsstrategie für Fahrzeuge mit Dieselmotor.* PhD thesis, Faculty of Mechanical and Process Engineering, Technical University of Kaiserslautern, Germany, December 2004.

[24] P. W. Kahan. *Lecture note: IEEE Standard 754 for Binary Floating-Point Arithmetic.* University of California, Berkeley, USA, 1997.

[25] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method.* Siam, 2003.

[26] K.-J. Langeheinecke. *Abgasnachbehandlung: Modellierung und Simulation.* Internal report from MDD33, IAV GmbH, Gifhorn, Germany, May 2005.

[27] K.-J. Langeheinecke. *Numerical solution of Selective Catalytic Reduction (SCR) model.* Internal report from MDD33, IAV GmbH, Gifhorn, Germany, 2007.

[28] J. M. Martinez. *Practical quasi-Newton methods for solving nonlinear systems.* Technical report, Institute of Mathematics, University of Campinas, Brazil, January 2000.

[29] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra.* Siam, 2000.

[30] M. Minoux. *Mathematical programming: Theory and Algorithms.* John Wiley, 1986.

[31] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 1999.

[32] M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic.* Siam, 1999.

[33] W. H. Press, S. A. Teukolsky, S. A. Teukolsky, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[34] F. Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer, 2006.

[35] Y. Saad. *Iterative Methods for Sparse Linear Systems (2nd edition)*. Siam, 2003.

[36] T. Schmidt. *Kurbelwellen-Engerie-Modell*. Internal report from MDD34, IAV GmbH, Gifhorn, Germany, July 2004.

[37] T. Schmidt. *Optimierung: Model Predictive Control*. Internal report from MDD34, IAV GmbH, Gifhorn, Germany, July 2005.

[38] G. Stiesch. *Modeling Engine Spray and Combustion Process*. Springer, 2003.

[39] I. Tkatchenko. *NOx Trap Modeling for Diesel Engine*. Technical report, Faculty of Mechanical Engineering and Marine Technology, University of Rostock, Germany, January 2006.

[40] S. Ulbrich. *Manuskript zur Vorlesung: Nichtlineare Optimierung III*. Zentrum Mathematik, University of Munich, Germany, 2004.

[41] R. van der Pas. *Memory Hierarchy in Cache-Based Systems*. Technical report, High Performance Computing, Sun Microsystems, Inc., November 2002.

[42] Q. Yi. *Applying Data Copy to Improve Memory Performance of General Array Computations*. Technical report, Department of Computer Science, University of Texas at San Antonio, USA.

[43] Y. Zhang, R. Tapia, and L. Velazquez. *On convergence of Minimization Methods: Attraction, Repulsion and Selection*. Technical report, Department of computational and applied mathematics, Rice university, USA, March 2000.

[44] U. Zimmermann. *Manuskript zur Vorlesung: Nichtlineare Optimierung*. Institute of mathematical optimization, Technical University of Braunschweig, Germany, 1996.